ISSN 0265-2919

80p 27

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

APPLICATION

MADE TO MEASURE Ergonomics is more than the study of the physical environment in which a computer is used: it also incorporates considerations of the way we interact with complex software — 521

SOUNDS IN SEQUENCE Our series on electronic music continues with a look at the MIDI interface — 534

HARDWARE

GOOD AT GAMES The Sega SC3000H is an inexpensive home micro from a company famous for arcade games machines — 529

COMPUTER SCIENCE

TURNING TURTLE By drawing a variety of shapes using turtle graphics, we introduce you to some of the basic commands of the LOGO language — 532

JARGON

FROM ELECTROSTATIC PRINTER TO EXPERT SYSTEM Our weekly glossary — 528

PROGRAMMING TECHNIQUES

GUIDELINES We consider the three main user aids available to the programmer: instructions, 'help' pages and 'signposts' — 526

MACHINE CODE

REGISTER TO REGISTER We investigate the variety of registers used by the 6809 processor, and consider some of the elementary instructions that they use — 537

PROFILE

FOSTERING AN IMAGE Australian creativity is behind the image-conscious software company, Melbourne House — 540

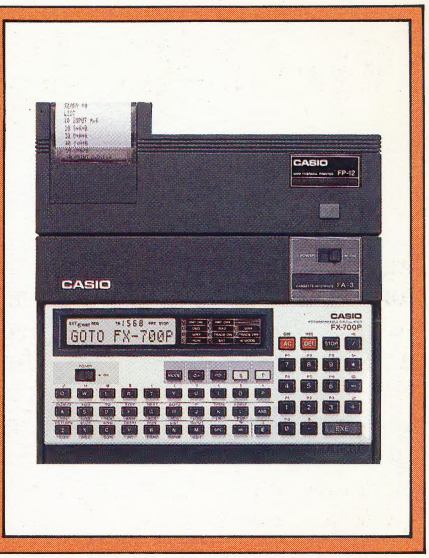
WORKSHOP

PROTECTIVE GEAR Our practical course continues by showing you how to build a buffer box — a device that protects the delicate internal mechanism of a computer — 523

THE WORKSHOP ANSWERS BACK We supply answers to the workshop exercises given last week — INSIDE BACK COVER

Next Week

● Pocket computers offer desk-top programming power at calculator prices. We examine the latest machines from Casio.
● Our LOGO series introduces procedure definition and editing — the essential features of any programming language.
● The equipment to interface computers to electronic musical instruments is available now for many home micros. We continue our investigation of modern musical techniques.



QUIZ

1) What was the first version of LOGO to appear on microcomputers?
2) Why is the 6809 wrongly called a 'pseudo 16-bit processor'?
3) What is 'fax'?
4) What is SQL?

Answers To Last Week's Quiz

1) The 'kludge' is a black box on the Sinclair QL containing extra operating system chips.
2) An IDC is an 'Insulation Displacing Connector'.
3) The advantage of digitised sound is that it can be reproduced by a synthesiser as part of a programmed sequence with variable rhythm, pitch and timbre.
4) An emulator is a computer system that mimics the operation of another, different, computer.

Editor Jim Lennox; **Managing Editor** Mike Wesley; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Researcher** Melanie Davis; **Contributors** Geoff Bains, Harvey Mellor, Mike Curtis, Steve Colwill, Steve Malone, Rory Forsyth, Richard Pawson, Graham Storrs; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

COVER PHOTOGRAPHY BY PAUL CHAVE VALIANT TURTLE COURTESY OF VALIANT DESIGNS LTD, LONDON

MADE TO MEASURE

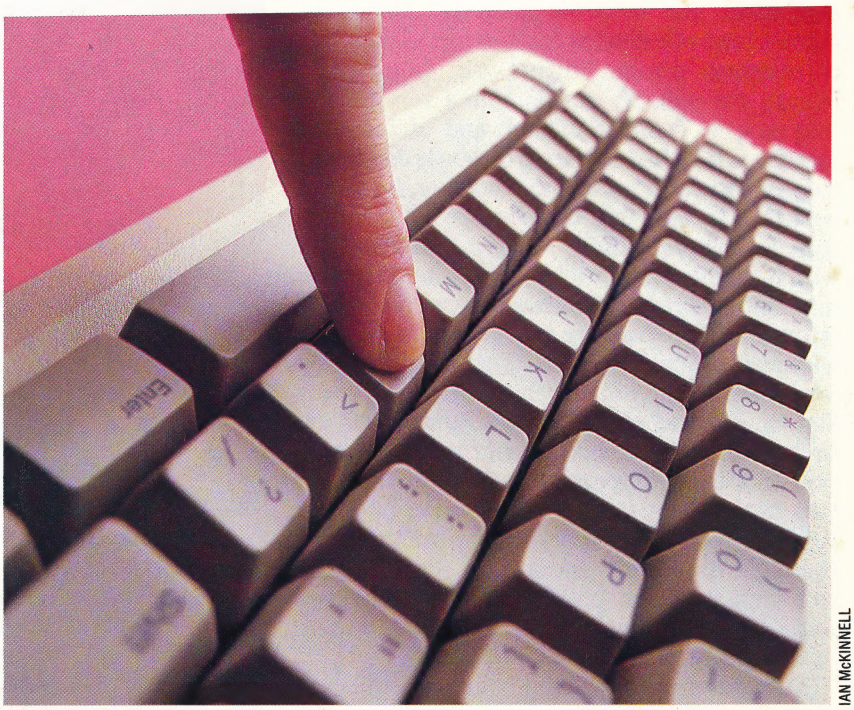
Ergonomics is the science of fitting the working environment to the human worker and draws upon such diverse disciplines as engineering and psychology, the physical sciences and physiology. It aims to discover the principles by which man-machine systems can be designed to operate efficiently while preserving the operator's health and comfort.

The arrival of personal computers in the home and workplace makes the implications of ergonomics relevant to us all. Colour and sound are good examples of how people's surroundings affect their performance at work. Some telephone switchboards relay music to callers on hold, to soothe and pacify them while they wait. A green VDU screen is found to be pleasant and restful by most computer users, whereas the more conventional white on black seems harsh and stressful after a few hours' use.

Different colour combinations provoke markedly different responses: blue on yellow is generally thought an attractive display, while cyan on green is unpopular. The colour of a room may affect its users' moods: yellows are thought to be 'cheerful', blues and greens are 'relaxing', and the typical office colour scheme of brown and grey is 'dismal'.

Such minor differences in preference may seem unimportant or irrelevant to the home micro owner, but ergonomists have repeatedly shown that when people are dissatisfied with their surroundings (often without knowing that it is the colour scheme or the background noise that is affecting them), they are more prone to eyestrain and backache, and are more likely to take time off for illness. Computer users commonly suffer from thoughtless room layouts and job structures. A major ergonomic study of one office where operators worked at terminals arranged around the edge of the room blamed their low efficiency and high error rate on the social isolation and stress caused by the layout. They found that when the workers were placed face to face at low-profile work stations in the middle of the room the whole atmosphere was lightened, and the quality of the work improved significantly.

The introduction of computers has often had a 'de-skilling' effect on jobs, making people's work undemanding, routine and uninteresting. It is now a recognised part of the systems analyst's task to make sure that the jobs allocated to people after a computer has been installed are satisfying and sufficiently demanding. In this analysis the expert advice of the ergonomist is needed.



IAN McKINNELL

Keys To Success
In general, the user interface starts at the keyboard, which has been subject to many ergonomic improvements — from sculpted keys and dished keyboards to numeric keypads and LCD displays. But the keyboard's essential failing — the inefficiency of the QWERTY layout — seems ineradicable, because most people are reluctant to learn new typing patterns. Such typically human emotions and apparent irrationality are often the major obstacles to human factors engineering

A truly ergonomic system design considers the human operator as an important system component with its own operating characteristics and tolerances. Humanitarian considerations and financial logic both acknowledge that people's feelings, perceptions and habits are at least as important to the efficiency of a system as its address bus capacity or its clock rate.

The programmer can also benefit from careful study of the particular problems and demands of his job. Programming requires constant care, logical method and painstaking attention to detail, yet few people exhibit these disciplines naturally, and most resist their imposition. This is one reason why bug-free programs are impossible to guarantee.

Applying ergonomics to the design of new programming languages and system design methods is a fascinating technique, offering many benefits to programmers and their employers. Psychologists have spent years studying people as information-processing devices, and have clear (though as yet incomplete) ideas of the brain's memory structures, speeds and capacities. They can help in designing program control and data structures that people are comfortable with and can use naturally or with little training.

Psychological considerations are a long-established part of training, selection and organisation methods within industry, too. Most significantly, psychologists can ensure that

designers concentrate first on making the system fit the people, rather than assuming that the user will adapt to the system.

In recent years, ergonomists have been studying the way that people react to complex software — the so-called 'user interface'. Through most of the history of computing, the users have been skilled, highly motivated professionals prepared to accept discomfort and inconvenience and willing to attain a level of technical competence as the price of computer power. Today's user, however, is literally the ordinary person with a very limited tolerance of temperamental and demanding machines; if he had to learn a database command language to use a bank cash dispenser, he would be likely to take his business to the building societies instead! And it is not just casual users who have interface problems. Database systems that put megabytes of management information on every office desk are everywhere underused and misused, because extracting the information from them requires fluency in complex languages like SQL. Researchers and users alike hope that the next generation of natural language 'front end' systems will make it possible to communicate with the database or the financial model in plain English. In this type of system, the front end will translate the user's input into the system's command language, possibly prompting the user to specify or amplify his requests, and explaining or expanding the system's responses.

There are many ways to help the user, including the provision of 'help' facilities (see page 526). Artificial intelligence research has led to the development of knowledge-based programs that can explain how they arrive at decisions, and it is thought that these techniques could help to create 'adviser modules' to help and prompt program users. This type of 'intelligent' software raises a philosophical question, however: what is a good explanation, and to whom? The programmer might want an explanation of the data structures and the processes involved in a system, while the business user would prefer a discussion of means to achieve a commercial end. Such distinctions are semantic and linguistic problems for the ergonomists and the computer scientists.

A similarly grey area in the study of the user interface is known to psychologists as 'modelling'. People use mental models (such as national or racial stereotypes) as a way of filling in gaps in their knowledge; most of us would confidently predict a stranger's appearance, views, personality and politics simply from knowing his job — civil servant, say, or conductor (bus or orchestra). Similarly, people approach computers with stereotyped ideas about the machines' power and behaviour, and may see them as more intelligent and knowledgeable than humans performing the same sort of task. When they meet the kind of curt, impersonal responses that many software packages make (especially to incorrect inputs), they may consider the computers rude and unfriendly, even hostile — judgements that are, of course, totally inappropriate to computers. This personalised perception of the machine affects the whole interaction, often leading to inappropriate responses on both sides of the console.

Programmers attempting to instil user-friendliness can exacerbate the problem by introducing features that make the program seem more intelligent than it really is. For example, using cheery prompts like HELLO JOHN, I AM THE SPIRIT OF THE DATABASE may encourage the user to reply in kind, often with catastrophic results and consequent discouragement.

True user-friendliness needs a rather more skilful approach. It means thinking and caring about people, and allowing for their complicated reactions to computers and work; there is more to ergonomic design than tilting the screen and painting the disk drives lilac!

Working Conditions
It is generally acknowledged that people at work are influenced in their performance by gross physical factors such as worktop heights, air temperature and noise levels. However, the subtler effects of colour, light and perception of space are now being recognised as equally important aspects of the work system, especially where computers are used. A systems analyst installing a new system must consider all these factors, in addition to choosing the hardware and software

Flexible FORTH

People can be frustrated and constrained by received ideas just as surely as by the design of their physical environment. FORTH, the language seen by many as a replacement for BASIC, was invented by astronomer Charles Moore at Kitts Peak Observatory in Arizona. Moore was working on controlling telescope movements using FORTRAN programs, but found that language too biased towards pure processing and unsuitable for external control applications. Writing a new language was his solution to the problem. FORTH differs from rigidly structured languages like FORTRAN in allowing the user to create virtually a new dialect of the language to suit each new programming task. The price of FORTH's flexibility is its starkly unfriendly approach — an efficient system isn't necessarily a comfortable one to work with

PROTECTIVE GEAR

So far in the course, we have looked at how the user ports of the Commodore 64 and BBC Micro work, and designed some simple input programs. In this article, we investigate ways of producing an output from the user port and outline a comprehensive control network that we will show you how to build in future instalments.

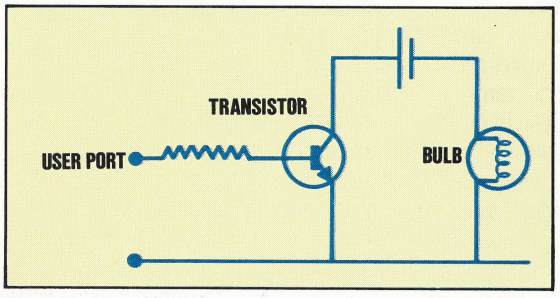
In most microprocessor applications the term 'buffer' (see page 208) has come to mean a temporary storage place for data that is being transferred from one part of a computer system to another. In analogue electronics, however, the term is used to describe a circuit that protects one device from the actions of another. If we wish to connect and drive electric motors or other electrical components under control from the user port then we must protect the delicate internal circuits of the micro from the components that we attach to it.

The input/output chip inside your microcomputer works at voltage levels of 0 and +5 volts and uses currents measured in a few milliamps (mA). Therefore, we must ensure that we don't put voltages higher than +5 volts on any of the user port lines, nor draw more than about 30 to 40 mA of current.

In our introductory instalment of this project (see page 514) we showed you how touching the bare wires of the user port lead together could change the contents of the data register. We found that earthing the register cells in this way didn't introduce dangerous voltages or currents to the system, and therefore no protection for the micro's internal circuitry was required. If we wish to connect other devices, however, then protection must be included, and this can take several forms. We may wish only to ensure that no more than 20 mA of current is drawn from any one port pin. This could be done by using a relay connected to the user port to switch the output device on and off, and inserting a resistor in the feed circuit to the relay. If the circuit operates at +5 volts, and we require a current of not more than 20 mA, then we can use Ohm's Law (voltage = current × resistance) to calculate the resistance value needed:

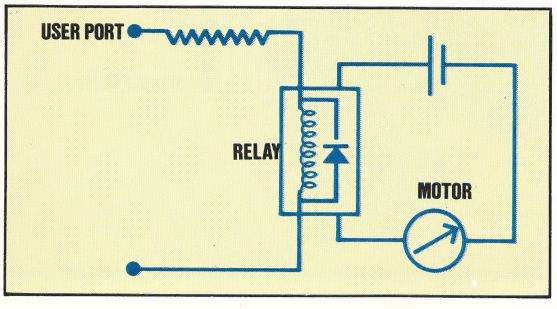
$V = I \times R$
$R = V / I$
$R = 5 / 0.02$
$R = 250$ ohms

Alternatively, the output from a user port pin could be used to trigger a transistor switch to complete an external circuit:

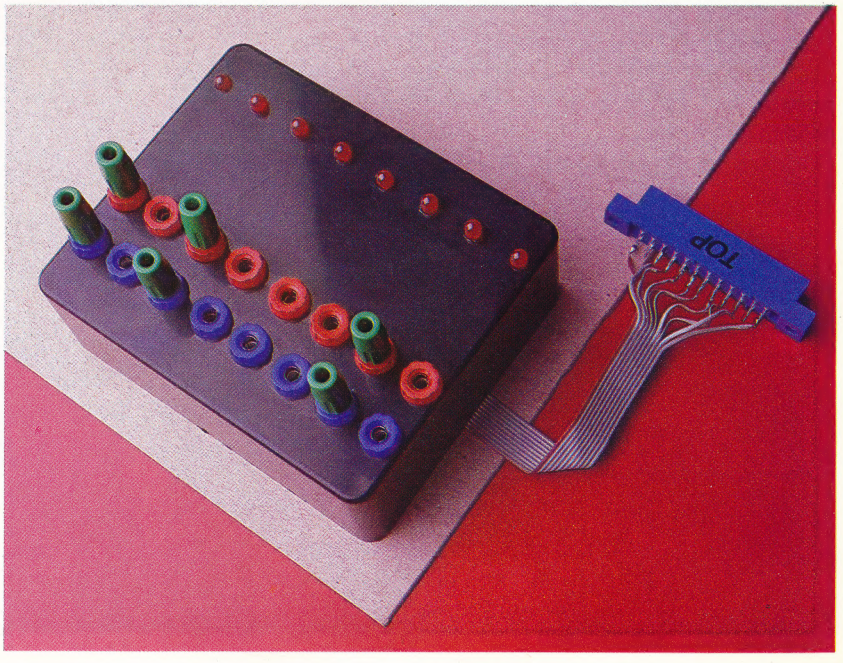


The buffer box that we will construct uses the transistor switching principle to protect the user port. This is a matter of convenience, as the circuits for all eight lines are available on a single chip.

Once buffering of the user port has been achieved we can go on to add a series of modules to the user port, which will allow us to connect other devices to it. These modules will make it possible for us to control the switching of LEDs, low and high voltage motors and mains relays. We will then be able to control household devices such

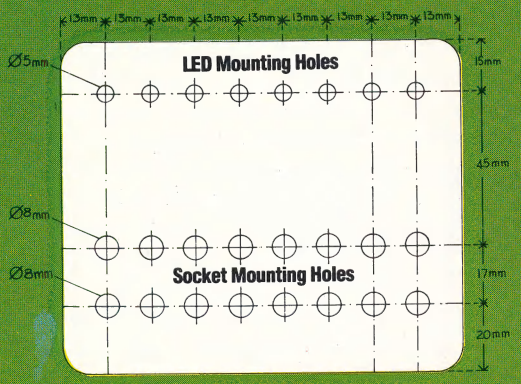


LIZ DIXON



The Go-Between
The buffer box is connected to the user port of the Commodore 64 or BBC Micro by the lead described in the last instalment (see page 514). The box protects the computer against harmful input/output current levels. The LEDs indicate the state of the user port output lines, and the plugs and sockets act as on/off switches on the input lines.

Drilling The Box



The LEDs and black and red sockets are mounted in the base of the box as shown in the template. The plastic of the box is soft and accepts a drill easily, but, to prevent skidding and scratches, cover its face with strips of masking tape, then mark out the drill holes. Start the holes with a bradawl or fine drill, then drill the holes — LED holes 4mm, sockets 8mm in diameter

The Naming Of Parts

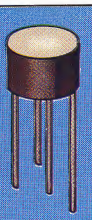
To construct the interface unit you will need the parts listed here. We give the Maplin part numbers, but you may get them (or reasonable substitutes) from any electronic components supplier. The exact dimensions of the case are not critical but our design exactly fits the box described.

Quantity	Item	Maplin No
8	4.7 K-ohm 0.4 Watt resistor	M4K7
8	240 ohm 0.4 Watt resistor	M230R
1	1 µF electrolytic capacitor	FF01B
1	0.1 µF capacitor	BX76H
8	Red LED	WL27E
8	1N4148 diode	QL80B
1	W005 bridge rectifier	QL375
3	7407 hex buffers	QX76H
1	µA7805UC voltage regulator	QL31J
1	36 strip × 50 hole veroboard	FL09K
3	14 pin DIL chip socket	BL18U
1	2 oz roll 22 swg tinned wire	BL14Q
1	metre 12-way ribbon cable	XR65V
8	Black 4mm socket	HF69A
8	Red 4mm socket	HF73Q
8	Black 4mm plug	HF62S
8	Red 4mm plug	HF66W
1	2.1mm PC mounting power socket	RK37S
1	12-way minicon socket	YW30H
1	115 × 95 × 37mm plastic box	LH22Y

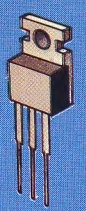
New Parts

The parts list contains many components already described in the course (see pages 115 and 138), but the parts we have illustrated may be unfamiliar. The power socket, rectifier and regulator are part of the box's power supply. Our design will allow us to use almost any mains transformer as the input, provided that its output is between 7v and 25v — AC or DC.

The 12-way socket will be the box's input/output port through which data is passed to and from external devices.

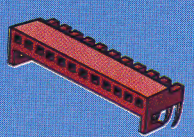


Bridge Rectifier
Converts a range of input voltages (AC or DC) to DC voltage of known polarity



Voltage Regulator
Smooths the output of the bridge rectifier (which may be 'pulsed' or 'ragged') to a steady 5v DC

Power Socket
Accepts a 2mm power plug, as used on many computer PSUs (the Spectrum, for example), and mounts directly onto the circuit board



Minicon Socket
Mounts directly onto the board and allows easy connection of external cables

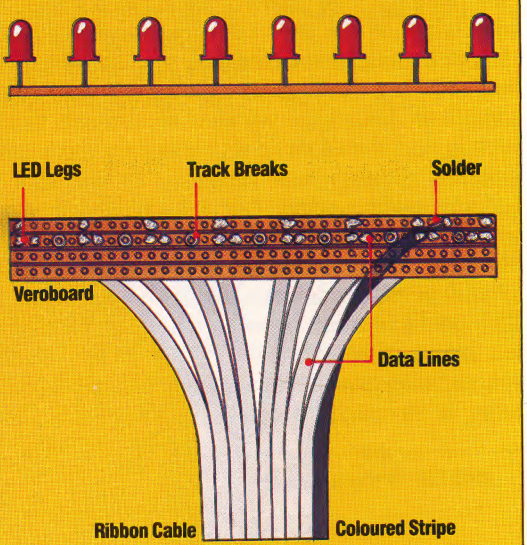
as lights, tape recorders, televisions and so on. In addition, we can add a digital-to-analogue converter, which will then enable us to drive decoded seven-segment displays. Because of the low voltage and current output of the user port we will also need an external power supply of nine volts. As each module is constructed it will be connected to a common bus, along which the eight data lines, an earth and a nine volt power line will be routed. In this way, we can 'piggy-back' or interconnect modules to the system. This then, is our plan of action for the forthcoming instalments of the Workshop course.

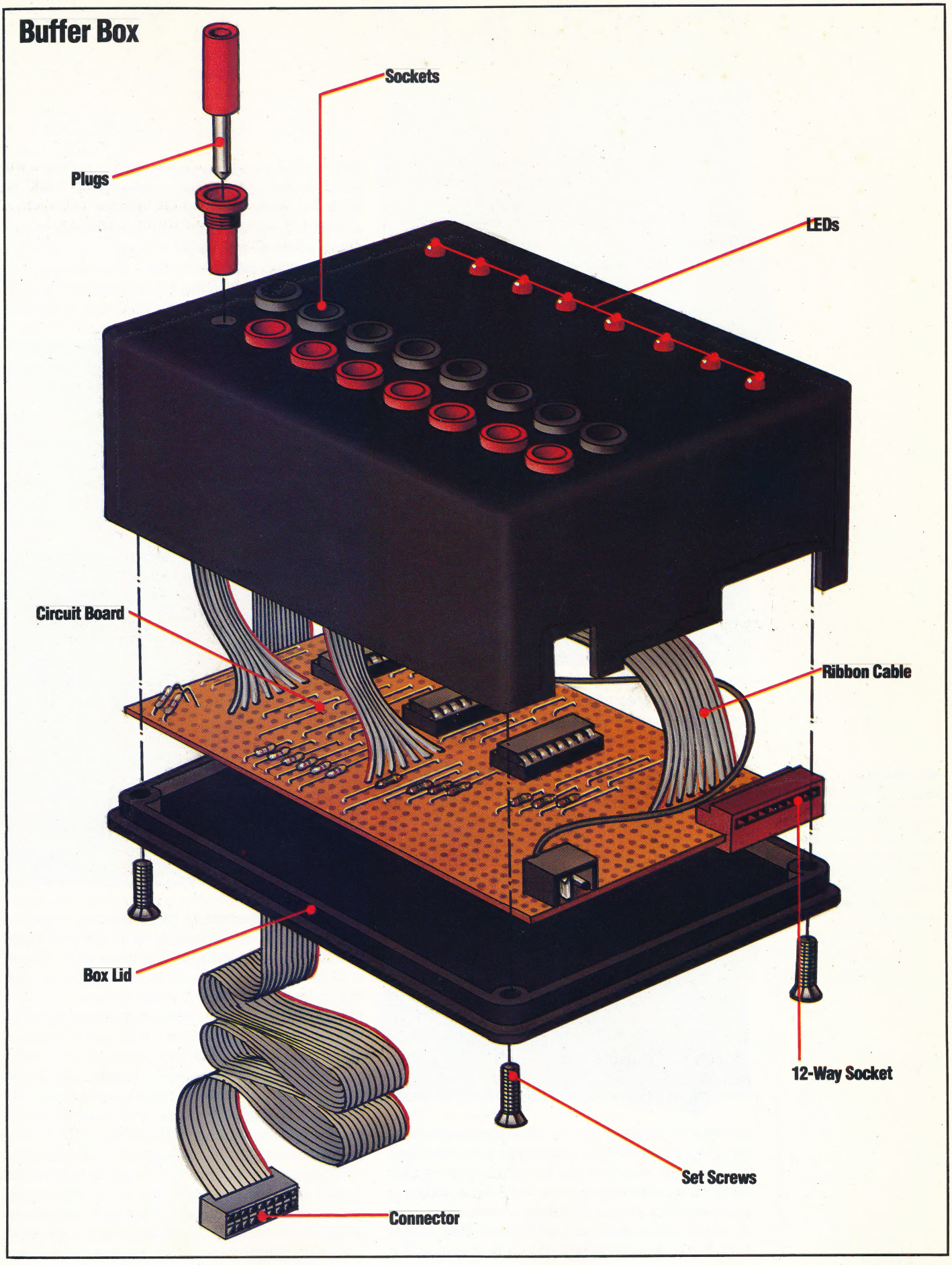
Building The LED Display

The LEDs will occupy a strip of veroboard four tracks wide, each track having 36 holes. Insert the LEDs as shown, with the longer legs on the edge track, allowing four holes between each. This should be the same as the spacing of the drill holes in the box; if necessary, reposition the LEDs. Solder the legs to the copper tracks, being careful not to run solder from one track to the other. Use a multimeter to check the resistance between the two tracks; if it is zero, you have bridged the tracks somehow.

Cut a 20 cm length of the 12- way ribbon cable, and remove three wires, leaving a nine-way ribbon including the coloured wire. Bare and tin the ends of the wires. Solder the coloured wire to the edge track of the board. Now solder each of the remaining wires along the other track, each wire next to an LED leg. Cut this track in seven places, so that each pin and wire pair is isolated on its own little strip of track. Once again, test for bridging between tracks and across the breaks.

Gently ease the LEDs and board into the holes in the box, screw the sockets into their holes, then sit back and admire your work while waiting for the next instalment, in which we will show you how to build the circuit board





ROSALIND BUCKLAND

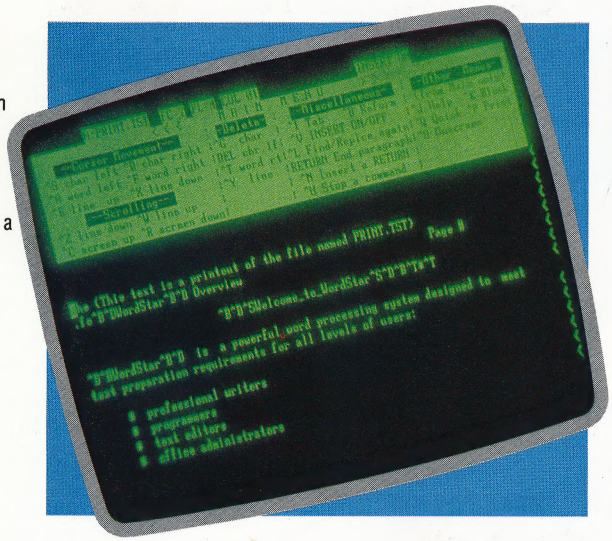
GUIDELINES

As more memory becomes available on microcomputers, the techniques used to guide the user through the workings of a program can become more sophisticated. Here we discuss the design and implementation of general-purpose 'help' routines that may be incorporated into your own programs.

Memory is now cheap. The next generation of home computers, which may have a *minimum* of 128 Kbytes of RAM, will leave most of us with far more memory than even our most ambitious programs will ever require. Throughout the history of computing, a shortage of memory has been the major excuse for failing to provide users with sufficient instructions, sensible error messages or on-line help. Now there is no excuse.

There are three main user aids that can be provided within a program: instructions, 'help' pages and 'signposts'. Instructions take two forms. They can be given in a single block at the beginning of the program, or they may be supplied as required throughout the program (as prompts for user input, for instance). Ideally, both should be available to the user.

In their simplest form, instructions may simply be a page — or several pages — of text explaining in clear English how to use the program. The text



Words Of Advice
Micropro Wordstar provides a top-selling example of command-driven software with 'on-line help'. The on-screen help menu can be abbreviated or removed by the user, but an enormously detailed Help file structure is always available at a single keypress

can be held in strings or DATA statements within the program, and will be displayed when required by a call to a subroutine written for this purpose. At the start of the main program, the user is asked if instructions are needed; if they are, the subroutine is called. Thereafter, other routines that accept user input should be tailored so that a specified input ('?' is common, or you could use 'I') triggers a call to the instructions subroutine. It is a good idea to create a standard 'display instructions' command, and modify any library input routines to accept it. Don't forget to modify any prompts used in your routines so that 'Press any key for more...' becomes 'Press any key for more (or "I" for instructions)'. This will give you a standard format that will be used in all your programs.

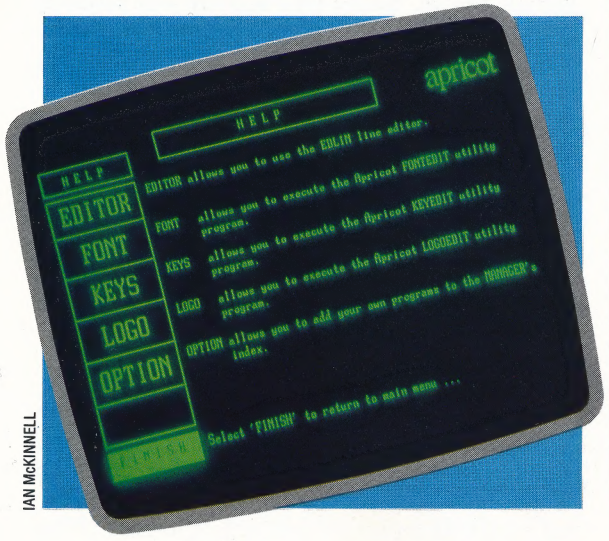
But instructions need not be text-only. Diagrams may be included, and the instruction routines can be developed to give examples and allow the user to practise and learn. Such instruction routines are common in programs that run scientific experiments — here the user may be required to perform a specified task to a particular level of skill before being allowed to progress to the main program. Such 'teaching' routines are not easy to develop because they must simulate the

behaviour of the rest of the program, as well as evaluating the user's performance. It is well worth the attempt, however, as designing this type of routine will give an indication of the problems that the main program will present to the user.

In a similar fashion, 'help' pages may be called up to explain the operation of particular parts of the program. This facility is found in many systems, where it is available to explain the use of commands — the Unix operating system, for example, allows the entire user manual to be accessed as on-line help! Providing help in your own programs need be no more difficult than supplying instructions: at each appropriate point, simply allow the user to enter a help request instead of the usual input — when this happens the program should call the relevant help routine. A complex program is likely to require a large

number of help pages, so again a general help routine is desirable. This may require the user to input a number to identify the particular help page required. On disk-based systems, the help pages may be held on disk as separate files. The help routine will then create the appropriate file name from the user's input, read in the file and display it on the screen.

Both help and instructions routines may well take up more than just a single page of information. If this is the case, your display routine should be designed in such a way that the user is able to move backwards and forwards through the pages at will. You should also ensure that the user can leave the routine at any stage and return to the exact point at which the main program was left — it is very frustrating to go through 10 pages of redundant information each time the instructions are required! If a prompt had been given, it will now have been lost so it must be repeated. The help routine should set a flag that tells the calling routine that it must go back to the last instruction before the help call, first clearing the flag.


apricot
HELP
HELP
EDITOR
FONT
KEYS
LOGO
OPTION
FINISH
EDITOR allows you to use the EDLIN line editor.
FONT allows you to execute the Apricot FONTEDIT utility program.
KEYS allows you to execute the Apricot KEYEDIT utility program.
LOGO allows you to execute the Apricot LOGOEDIT utility program.
OPTION allows you to add your own programs to the MANAGER's index.
Select 'FINISH' to return to main menu ...

IAN McKINNELL

A common metaphor for user interactions with complex programs is to think of the user navigating through a tangled network of logic. The newcomer to the program will not understand its structure and can easily become disoriented and lost. Thus 'signposts' are needed to guide the user. A menu is the clearest example; this operates like a road sign that shows the possible exits from a junction. Systems such as Apple's Macintosh and Lisa work in a similar way, using icons instead of menu options.

Some directions are more important than others. In a command-based system, there may be dozens of possible commands. However, not all of these will be relevant or even possible at a given point in the program. If the number of options is small, it is useful to display a line or two to explain what they are. Some options — such as QUIT — must be available at all times, so it is a good idea to keep these on permanent display. UNDO, SAVE and other application-specific commands may also be constantly available. A common technique is to program such commands onto function keys and display a single-line message to show each key's function. It is always good practice to display a signpost that indicates the way out of a program — this instils confidence in first-time users, whose major concern is often to find the emergency exit!

Good Management

The ACT Apricot's Manager software guides the user through a suite of unfriendly utility programs by its hierarchical menu system. Help is an option on every menu, and consists of an explanation of the other menu items. This is a good example of classic menu-driven software supported by large Help files

Some experimental systems have been developed that can monitor a user's performance and adjust the level of help given accordingly. Commercial programs with this feature are still a long way off, but it is possible to use simple techniques to achieve at least a part of this goal. If the user is asked to give his or her name each time the program is run, then a file can be kept of users and their skill levels. These levels can be calculated (from the number of times a particular user has run the program, say, or from the highest score achieved if the program in question is a game) and updated at the end of the program run. As the skill level increases, the type of help and signposting supplied will change, becoming briefer and less intrusive. The user might also be asked to choose the level of help required, as in the Wordstar word processing package. Ideally, both alternatives would be used.

Incorporating help can be a valuable guide to improving a program's performance. Once a help routine has been designed (such as the one provided here) it is a simple matter to modify it to record which help pages were used and how often they were needed. This gives a clear indication of the trouble-spots in the program.

ELECTROSTATIC PRINTER

Electrostatic is a term that is often used as a synonym for 'electrosensitive', but it is better employed as a description of so-called 'laser' printers that are used in large computing installations. A laser printer is really a modified electrostatic photocopier. An electrostatic charge is applied evenly to the surface of a smooth zinc drum, and then scanned by a laser beam, which is switched on and off by the computer. Where the surface is exposed to the laser beam, the charge is neutralised. After the drum has been scanned, a sheet of paper is passed over it and acquires the electrical image. The paper is then covered in fine carbon powder. The carbon sticks only to the charged areas, and is then baked on by a heating element, producing an 'electronic painting' of the image to be printed. Laser printers can produce an entire A4 sheet of printing in a few seconds, irrespective of the amount of text on the page. These devices are falling in cost, but the cheapest is still around £5,000. Full-colour laser printers are expected to appear on the market soon.

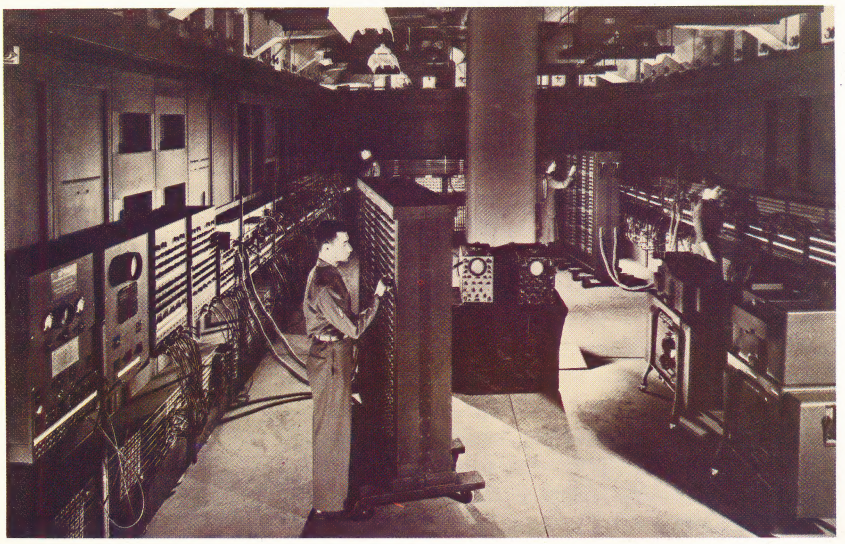
EMULATOR

An *emulator* is a computer system that mimics the characteristics of another computer. Microcomputer software, for example, is often developed on a mainframe to take advantage of the larger system's greater storage capacity and better debugging facilities. If, for example, Spectrum software is being developed, a Spectrum emulator must be written for the mainframe, which then behaves exactly like a Spectrum microcomputer.

ENIAC

An acronym for Electronic Numerical Integrator And Computer. ENIAC was designed by John J Mauchly and J Presper Eckert Jr at the University of Pennsylvania during the years 1943-1946. It was a general purpose electronic calculator that used vacuum tubes (valves) as relays. ENIAC was originally designed to help with the creation of ballistics tables during the Second World War, although it was not completed until the war was over. Because of its immense size and the amount of heat generated by the electronic circuits, ENIAC had to be housed in a large, air-conditioned room.

ENIAC
This computer, for all its ponderous construction and the massive power drain of its vacuum tube logic, brought computing out of the preserve of electromechanics and into the electronic age



COURTESY OF THE SCIENCE MUSEUM

EOF

The *End Of File* indicator is a control character that is embedded at the end of a file of data to instruct the operating system to stop searching. It is mandatory on a cassette-based system, but is seldom used on disks because the directory file keeps a record of the length of all disk files.

ERGONOMICS

The science of designing machines to fit in with human beings, taking into account our physical and psychological attributes, is called *ergonomics*. Physical characteristics are considered when the external features of the system are designed — ergonomics is responsible for developments like detachable keyboards, adjustable screens, high-definition monitors, etc. Psychological considerations have led to an awareness of concepts such as 'user-friendliness', which is a term that indicates a particular program has been designed to cater for as many eventualities as possible.

EXCLUSIVE-OR

The Exclusive-OR operation is one of the fundamental building blocks of Boolean algebra. The operation requires two input bits and one output. The output is 1 if either of the inputs is 1, but is 0 if both inputs are 0 or both inputs are 1. The Exclusive-OR operation may therefore be thought of as a difference tester — the output will always be true (1) if the inputs are different. An Exclusive-OR gate may be constructed from two AND gates, two inverters and an OR gate. Its operation can be considered as:

Output = (A AND NOT B) OR (B AND NOT A)

Exclusive-OR is also available as a single operand in many microprocessors. It compares the contents of the accumulator with another specified byte, giving a 1 for each bit position where the two bytes differ.

EXPERT SYSTEMS

Computer programs written for very specific applications that previously required the presence of experts are called *expert systems*. In an expert system, the computer terminal acts as a temporary substitute for an experienced person. The expert system takes over detailed tasks and frees the human's time for more productive activity, or allows someone to be assisted at the times when a human expert cannot be available. Expert systems can be used for medical diagnosis, the detection of problems in mechanical and electronic systems, geological engineering, and computer-aided design.

GOOD AT GAMES

Sega is a Japanese company that is best known for its coin-operated arcade games machines, such as Frogger and Zaxxon. Given this background, a home computer from Sega should be an excellent games machine. The SC3000H fits that description very well.

Sega's new home computer, the SC3000H, has been shown at several recent computer exhibitions, where it has attracted favourable comment. Although hardly revolutionary in design or function, using as it does the now-familiar Z80A processor, it is a well-designed and easily expandable home computer with a wealth of available software. As yet, Sega has no UK distributor, but the SC3000H is expected to cost around £150.

An attractive, light machine weighing 1.1 kg (2.4 lb) the Sega has a black plastic case with white alphanumeric keys and grey operations keys (for special functions, Control, Shift, Return, etc.). It has moulded plastic typewriter-style keys, which travel about one centimetre when pressed. The keyboard has a decidedly 'clicky' feel to it, which is possibly a carry-over from the soft rubber keyboard used on the Japanese version. Overall, though, the quality of the keyboard is good for a machine in this price range. As well as the standard keys, the Sega has one non-programmable function key, which is used to enter BASIC keywords, a 'graph' key for accessing keyboard graphics symbols, 'clear screen' and insert/delete keys, and a four-key cursor cluster that is ideal for games-playing. However, a big drawback is the absence of graphic symbols on the keys. The 'soft-key' SC3000, which will not be distributed in this country, has symbols printed on the keys; without these, operating the SC3000H in BASIC is made more difficult.

The SC3000H is well-equipped with interfaces. Two Atari-style joystick ports are located on the left-hand side, there is a ROM cartridge slot on the right, and connections at the rear of the machine include output to a television set, a composite colour monitor port, DIN-type printer port, cassette interface and power socket for the nine volt mains adaptor. Also included is a switchbox for use with a television set and a BASIC cartridge with a small instruction booklet.

Setting up the Sega is straightforward, which is just as well as there is little in the way of documentation to help. A two-page leaflet describes the connection of the computer to a television set and to the power supply. A green

CHRIS STEVENS

Nippon Newcomer
Planned to sell at about £150, the Sega SC3000H is intended to provide competition for the Sinclair Spectrum and Commodore 64. The machine has a full range of peripherals, including cassette recorder, joysticks, and colour printer/plotter

LED indicates that the machine is switched on, but the leaflet fails to mention that the Sega will not function unless a cartridge, holding either BASIC or a game, is inserted into the ROM port. There is no internal BASIC interpreter, and once the BASIC cartridge has been loaded the SC3000H is left with a derisory 515 bytes of user RAM — less memory than provided by the unexpanded ZX81. This is a distinct disadvantage in the marketplace, as it means that the user will need to purchase an expansion module to use the machine for anything other than simply running games cartridges.

The manual may also lead to confusion as it refers to the 'soft' keyboard of the SC3000 and necessitates the user consulting the printed keyboard diagram to locate any required graphic symbol. This fault may be corrected on later versions. The keyboard diagram also shows BASIC keywords printed on or above many of the keys, Spectrum-style. These are accessed by holding down the function key at the same time as the relevant alphanumeric key. Commands such as RUN, LOAD, and GOTO, mathematical functions (ABS, SIN, COS, TAN) and string functions such as LEFT$, RIGHT$ and MID$ may all be accessed in this way, but again constant reference to the manual is necessary to find the relevant keys.

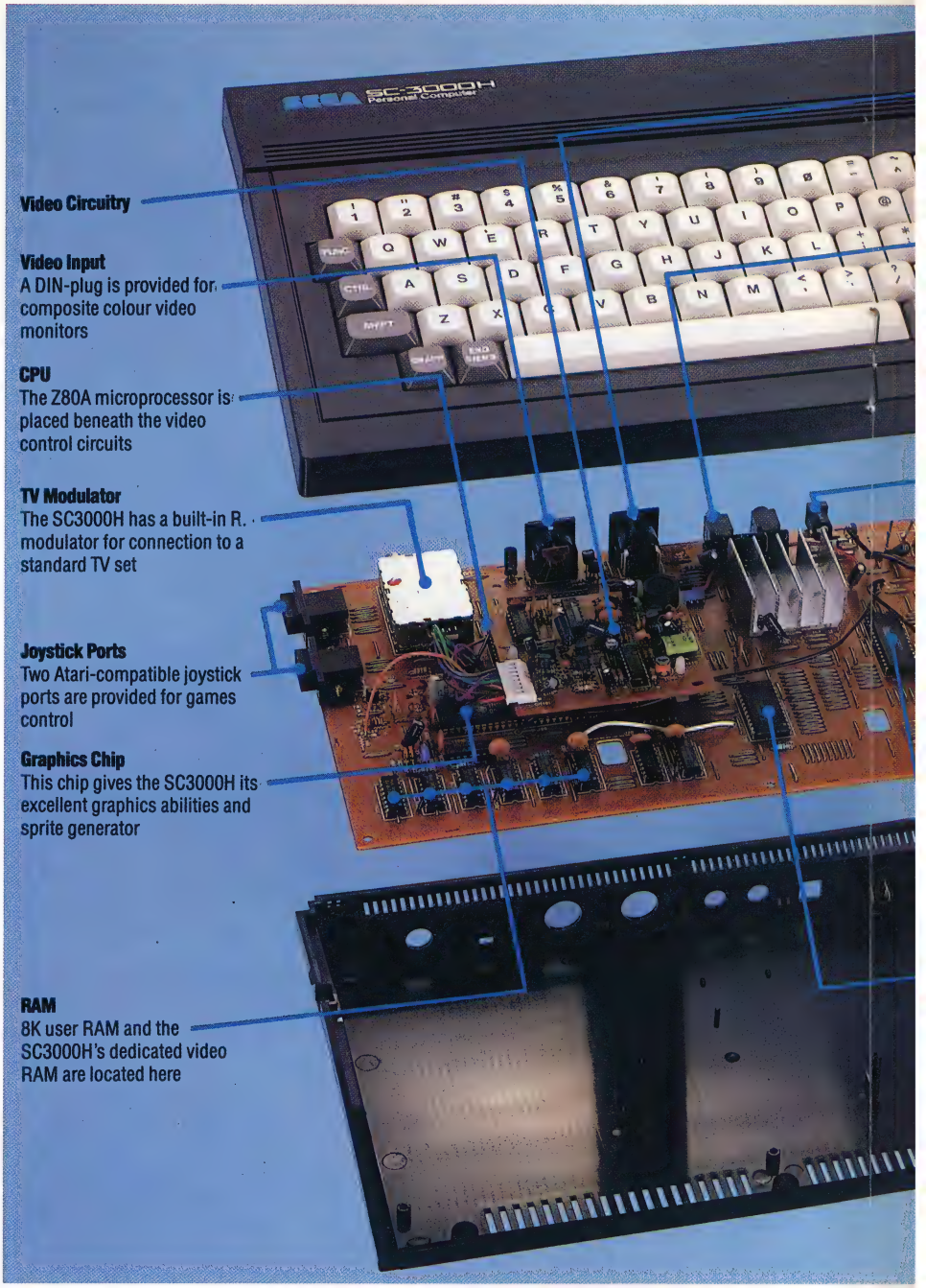
In operation, the SC3000H graphics are very impressive. The display features two screens — the text screen gives 24 rows of 40 columns in two colours, while the graphics screen has a resolution of 256 × 192 pixels and can display 16 colours. Brightness levels may be adjusted to give up to 210 shades, and up to 32 sprites may be created. The Sega BASIC, which is similar to Microsoft Extended BASIC, uses DRAW, COLOR, PAINT and SPRITE commands for programming graphics.

The SC3000H also has six sound channels, which can be addressed via POKE commands or by using the BASIC commands BEEP and SOUND. A music cartridge aids the composition of melodies, although the tunes produced fall short of the 'synthesiser quality' claimed in the manual.

Sega's arcade experience is reflected in the quality of its games software. Game graphics are generally good, although the SC3000H appears to have difficulty in displaying both text and graphics at the same time. Visually, the games are very much like their arcade counterparts, and the high-quality sound makes them all the more enjoyable. The Reset key is used in an unusual fashion: instead of restarting a game, it functions as a toggle switch to provide a pause in the action.

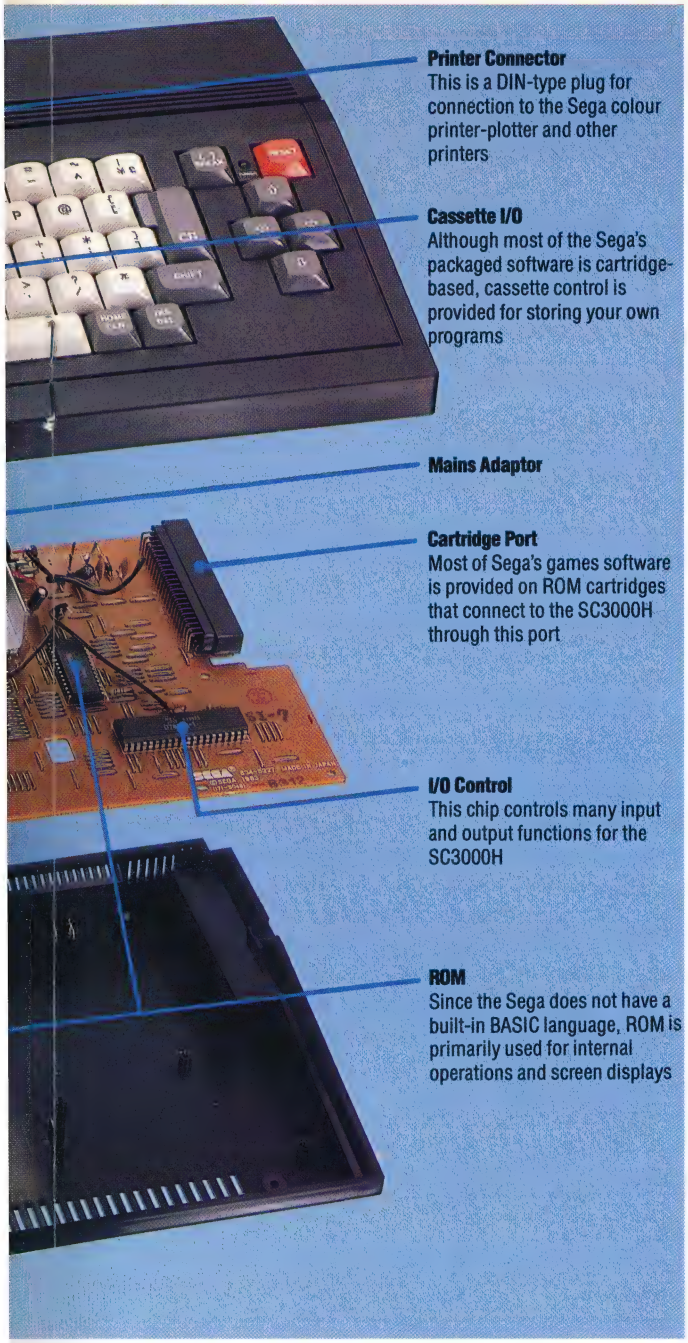
Sega supplies two different types of joystick, but neither of these provides fast, smooth movement. However, the sensibly designed cursor 'cluster' allows easy keyboard operation of games software.

The company markets a good range of peripheral equipment for the machine, including a cassette recorder, colour printer/plotter and an expansion unit. This gives an extra 64 Kbytes of RAM and has a built-in compact disk drive. The expected price of this unit is a reasonable £150.


Video Circuitry
Video Input
A DIN-plug is provided for composite colour video monitors
CPU
The Z80A microprocessor is placed beneath the video control circuits
TV Modulator
The SC3000H has a built-in R. modulator for connection to a standard TV set
Joystick Ports
Two Atari-compatible joystick ports are provided for games control
Graphics Chip
This chip gives the SC3000H its excellent graphics abilities and sprite generator
RAM
8K user RAM and the SC3000H's dedicated video RAM are located here

Sega SC3000H Joysticks
There are two different joysticks for the Sega, each suited to different playing styles. Both use an eight-position switch mechanism for stick movement and are compatible with the standard Atari nine-pin connector

CHRIS STEVENS



Printer Connector
This is a DIN-type plug for connection to the Sega colour printer-plotter and other printers

Cassette I/O
Although most of the Sega's packaged software is cartridge-based, cassette control is provided for storing your own programs

Mains Adaptor

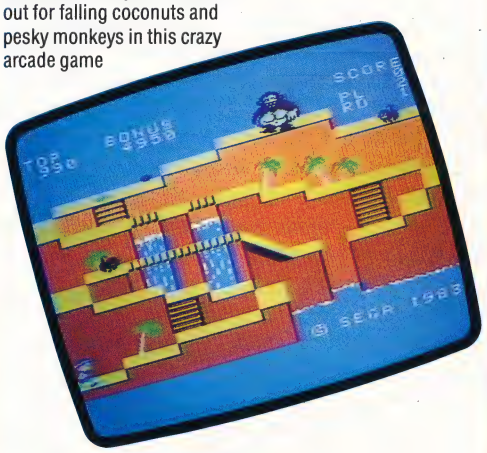
Cartridge Port
Most of Sega's games software is provided on ROM cartridges that connect to the SC3000H through this port

I/O Control
This chip controls many input and output functions for the SC3000H

ROM
Since the Sega does not have a built-in BASIC language, ROM is primarily used for internal operations and screen displays

Safari Hunting
Hunt fierce jungle animals with a tranquilising gun

Congo Bongo
Climb treacherous cliffs to chase after the gorilla. Watch out for falling coconuts and pesky monkeys in this crazy arcade game



Baseball
Go to bat against a range of opposition in this realistic version

Sinbad Mystery
A combination maze-adventure game

Sega Software
The software provided for the SC3000H has the kind of graphics and sound expected from an electronic games company like Sega. Many of the games have excellent '3-D' displays and innovative reworking of 'standard' games formats

SEGA SC3000H

PRICE
Approximately £150

DIMENSIONS
353×210×46mm

CPU
Z80A, 4MHz

MEMORY
6K RAM, expandable internally to 48K
18K ROM, expandable internally to 32K
16K video RAM

SCREEN
24 rows of 40 columns of text; graphics resolution of 256×192 with sprites and 16 colours in 15 brightness levels

INTERFACES
ROM cartridge port, joystick ports (2), TV and composite video, audio, cassette, and printer

LANGUAGES AVAILABLE
BASIC, LOGO

KEYBOARD
64-key typewriter-style with cursor rose, function key, graphics and special character keys

DOCUMENTATION
Two page set-up sheet and the manual that comes with the BASIC cartridge. This little booklet is fairly thorough, but you are not told that the machine does nothing until you load a cartridge. All documentation refers to the 'soft-key' version of the keyboard, rather than the keyboard on the UK machine

STRENGTHS
This is a pleasant machine with good graphics and sound, and well-equipped BASIC. For the price, it is a good games machine and beginner's micro

WEAKNESSES
The documentation is abysmal. 8K memory makes it virtually impossible to program. The keyboard should have special symbols printed on it

The Sega SC3000H is a pleasant machine to use, and contains some useful features that are unusual on a micro in this price range. Sega should take steps to improve the documentation, and the keyboard needs to be restyled to include keyword and graphic labels. The major drawback is undoubtedly the limited user memory; at present the machine is suitable only for use with Sega's games software — anyone wishing to write programs will need to expand the internal memory or purchase the expansion unit.

TURNING TURTLE

Our introduction to LOGO (see page 506) examined its development as an educational aid by Seymour Papert. Many versions of LOGO are now available and here we discuss ways in which the language's turtle graphics can be used to draw complex shapes with the minimum effort.

Abbreviations

Many LOGO commands have abbreviations: here are the ones for commands introduced in this part.

FORWARD	FD
BACK	BK
RIGHT	RT
LEFT	LT
PENUP	PU
PENDOWN	PD
PRINT	PR
FULLSCREEN	FS
TEXTSCREEN	TS
SPLITSCREEN	SS

The first version of LOGO to appear on microcomputers was MIT LOGO; this is now regarded as the 'standard' version and is produced by Terrapin Inc for the Apple and Commodore 64 machines. Logo Computer Systems Inc (LCSI) produces another version for the Apple, Atari and Spectrum computers, and LCSI LOGO for the BBC Micro should soon be available. There are other versions, but these two are the most widely available. Our example programs all use MIT LOGO; where there are differences between MIT and LCSI versions, these will be explained in the 'Flavours' box.

There is only one way to learn LOGO — by experimenting! We will suggest certain things for you to try, but the best thing to do is to solve problems that you have set for yourself.

Once loaded, LOGO is in 'immediate' mode and is ready to receive and obey commands. In most versions, these commands must be entered in upper case letters. Type DRAW and you will see that the screen is divided into two sections (this is called 'splitscreen' mode). The upper section is for the graphics; this takes up most of the display area and in the centre is the 'turtle', represented by a small triangle. The lower section is for text, and at the moment will simply contain the prompt '?'.

The turtle is an object that we can communicate with by giving it commands. Thinking of the turtle as an 'object' will make programming with it easier to understand. The most important things to be considered are the turtle's position, its heading (direction), and whether the 'pen' it carries is down (in which case it will draw a line as it moves) or up (in which case the turtle will move without leaving a trace). Typing DRAW positions the turtle in the centre of the screen, facing straight upwards with the pen down.

Now let's try giving the turtle a command:

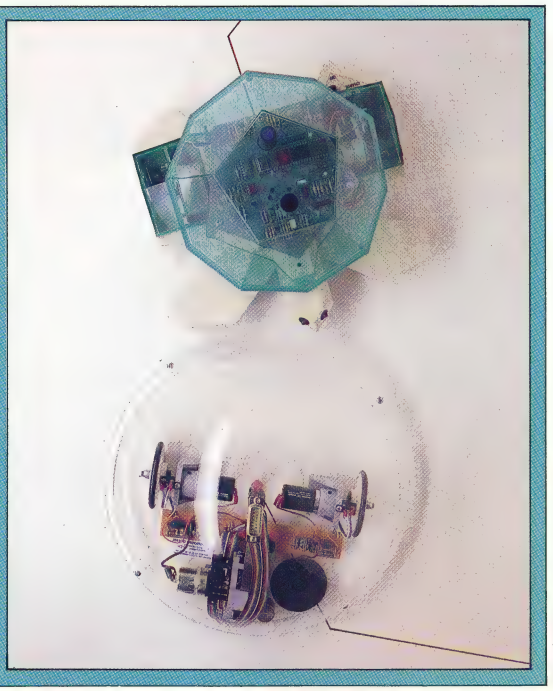
```
FORWARD 40
```

The turtle will move 40 units up the screen, drawing a line as it goes. FORWARD is a turtle command, and the number 40 is its 'input'. Some commands need inputs, while others do not — DRAW, for example, does not require an input.

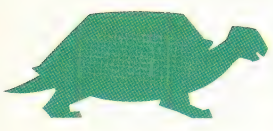
A second turtle command is BACK. BACK 10 instructs the turtle to move back 10 units. So FORWARD and BACK (each with a number of units as inputs) change the turtle's position on the screen. RIGHT and LEFT, on the other hand, do not change the turtle's position but simply rotate it — that is, they change its heading (direction). These two commands require an angle of between zero

Meet The Turtle

A turtle is a robot drawing tool. It has wheels, controlled by stepper motors, and a retractable pen. It can be instructed to move forward, back, left and right, and can lift or lower its pen. When lowered, the pen produces a drawing on the surface where the turtle is placed. When they were first developed, turtles were dome shaped, like the Edinburgh turtle shown here, and controlled from a computer keyboard. This turtle connects to the computer via a parallel cable and costs about £192. Newer turtles are remote controlled. A radio-controlled version of the Edinburgh turtle is now available for just over £200. And there is a turtle-shaped robot, the Valiant Turtle, also shown, which has an infrared connection to the computer. The Valiant Turtle costs about £229. By extension, the name turtle is also used to refer to the drawing cursor on the computer screen in LOGO. Most screen turtles are simple triangular shapes, although Atari LOGO displays a tiny turtle-shaped cursor



PAUL CHAVE



and 360 degrees as inputs. Experiment by using these commands — try drawing some simple shapes; see what happens if you instruct the turtle to move a greater distance than the screen size permits; try using negative numbers as inputs. To start again with a clear screen, type DRAW.

When trying out commands, you may find that the turtle moves into the text section of the screen, and thus appears to be 'behind' the text so that it can't be seen. These commands may help:

FULLSCREEN — allows the full screen area to be used for graphics;
TEXTSCREEN — removes all graphics and leaves you with text only;
SPLITSCREEN — returns you to splitscreen mode;
PENUP — allows the turtle to move without drawing a line;
PENDOWN — the turtle leaves a 'trail' as it moves.

The commands we have discussed so far are all instructions to make the turtle do your bidding. But you can also use LOGO to get information back from the turtle. The turtle's heading is measured in degrees; a heading of 0 indicates that the turtle is facing straight upwards, and the heading is measured clockwise through 360 degrees. To find the turtle's heading, type:

```
PRINT HEADING
```

The turtle's position on the screen is defined in terms of a co-ordinate system, with its origin at the centre of the screen (i.e. the turtle begins at a point with x and y co-ordinates of 0). You can find the turtle's position at any time by typing:

```
PRINT XCOR PRINT YCOR
```

Try out these commands by drawing a shape and then ascertaining the turtle's position and the direction in which it is facing. Use this data to return the turtle to its starting point.

You may have found that error messages have appeared in the text area during your experiments with LOGO commands. If not, then make a deliberate mistake to see what happens. For example, type:

```
FORWARD50
```

You will see the message:

```
THERE IS NO PROCEDURE NAMED FORWARD50
```

The reason for this is that LOGO requires a space between the command FORWARD and the input 50 so as to avoid confusion with a possible command called FORWARD50. You may also get an error message if you have typed a command in lower case.

LOGO is equipped with a line editor that enables you to correct instructions if you notice an error before you press Return. Use the cursor keys to move along the line to the incorrect text. To insert characters, simply type them in — the text to the right of the mistake will move automatically to accommodate any extra characters. The delete key removes the character to the left of the cursor. Once the line is correct, pressing Return will allow LOGO to accept the new instruction. If you have already pressed Return before noticing a mistake, typing Control-P will retrieve the last line for editing. This feature is equally useful for repeating commands.

Now we can try something a little more mathematical — for example, a square. Remember that a square has four equal sides and that its corners are right angles (90 degrees), so something like this will produce the desired result:

```
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
```

Notice that we can abbreviate the commands, and can put more than one command on a single line.

Unfortunately, you may hit a technical problem at this point — your square may look more like a rectangle. This is because of the 'aspect ratio' of your screen — that is, the ratio of the vertical step size to the horizontal step size. There is a LOGO command to deal with this: use .ASPECT followed by a number (the default is 0.8) to change the aspect ratio until your shape really is a square. Now try the following exercises: draw an equilateral triangle; a pentagon; a hexagon; various rectangles; a rhombus; a parallelogram — indeed, anything that takes your fancy!

You may simplify some of the commands and reduce the amount of typing by using REPEAT. To draw the square again, simply type:

```
REPEAT 4 [FD 50 RT 90]
```

REPEAT is a command with two inputs. The first is a number that indicates the number of times LOGO must do something, and the second is a 'list' of the commands to be obeyed. This list must always be enclosed in square brackets. So our square example tells LOGO that it must repeat the FORWARD 50 RIGHT 90 sequence four times. Now try using REPEAT to simplify construction of the shapes you have already created, and see if you can produce the star shapes shown in some of our illustrations.

Logo Flavours

On the Atari, the turtle looks like a turtle, not a triangle!
All LCSI versions:
Use CLEARSCREEN (abbreviation CS) to begin drawing.
Use Control-Y to recall the last line (except Spectrum, which does not have this feature).
To change the aspect ratio:
Apple LCSI — SETSCRUNCH followed by new ratio
Atari — .SETSCR followed by new ratio
Spectrum — SETSCRUNCH followed by two co-ordinate numbers [x y] (the norm is [100 100])

LOGO Exercises
Can you write procedures to create these shapes? The samples shown were drawn with LCSI LOGO on an Atari 600XL

Equilateral Triangle

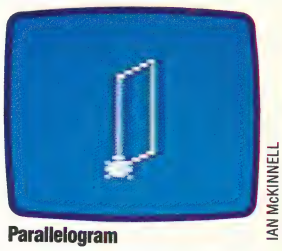
Pentagon

Hexagon

Rectangle

Five-Pointed Star

Ten-Pointed Star


Parallelogram

IAN McKINNELL

SOUNDS IN SEQUENCE

In the previous article in the series we saw how sequencing has become an essential part of the contemporary musician's repertoire. Sequencing is important, not only to co-ordinate the instruments of the musician playing live, but also in the studio where precise timing is vital in the mixing of sound. Now we look at a development of sequencing: the MIDI interface.

The sequencer has had a dramatic impact on music-making, in both live performance and recording studios. But the drawback of the sequencer is that it can control a range of variables within only one synthesiser. If a keyboard player has two synthesisers, one with powerful sequencing abilities and the other with good sound, there is no way to connect the two into a single co-ordinated unit. The problem is worse in studios that need to co-ordinate a variety of sounds from several different pieces of equipment. The purpose of a digital interface for musical instruments is to make this connection, and put control of the system in the hands of one, highly capable machine. MIDI is an attempt to do this in a standardised format so that any digital system can control another in the production of music.

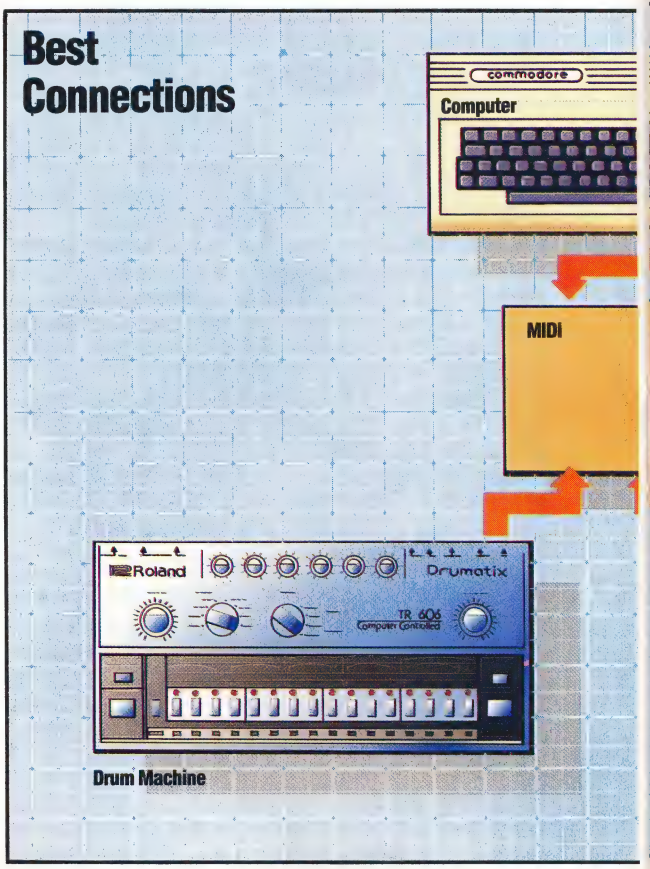
The MIDI (Musical Instrument Digital Interface) was developed after a series of meetings between the principal Japanese and American manufacturers of digital instruments. The present specification was finalised in August 1983. It is designed as a control circuit to transmit data from one instrument to another, or from a microcomputer to an instrument.

MIDI works on an eight-bit processor. Its designers had a choice between the two available means of transmission — parallel or serial. Parallel transmission provides individual lines so that the eight bits in each byte of data are sent simultaneously, and as a result permits a high transmission rate. Its disadvantage is the extra cost involved, and the inconvenience of having at least eight wires cabled to a 25-pin D-type connector. Serial transmission uses only two lines. On one line the data bits are sent one after the other; a second line is provided to enable the receiving instrument to signal parity errors (see page 187) in the data stream back to the master instrument or microcomputer. As a result, serial transmission is slower than parallel, but it has the advantage of a simpler connector and is considerably cheaper.

The original motivation behind MIDI was to provide a control circuit that could enhance synthesised music and provide some form of pitch control. Its price had to be within the range of most home computer owners and individual musicians using synthesisers and drum machines. Primarily for these reasons, MIDI transmission is serial.

MIDI transmits asynchronously, along the same lines as the RS232 interface that is standard for connecting many microcomputers to modems or serial printers. When data is transmitted asynchronously, each byte has to be defined for the receiving instrument. In MIDI, this is done by a Motorola 6850 ACIA (Asynchronous Communications Interface Adaptor) chip that adds two extra bits to each byte from the master instrument. The first is '0', the start bit, followed by the eight bits of serial data, and concluded by '1', the stop bit. This 10-bit serial word is then transmitted to the receiving instrument where a second ACIA chip converts it back to eight bits of real data.

The expensive ACIA chip is protected within the circuit by being opto-isolated. An opto-isolator is a device that uses photoelectric cells to allow two unconnected electrical circuits to exchange signals yet remain electrically isolated; voltage 'surges' will thus leave the chip undamaged.

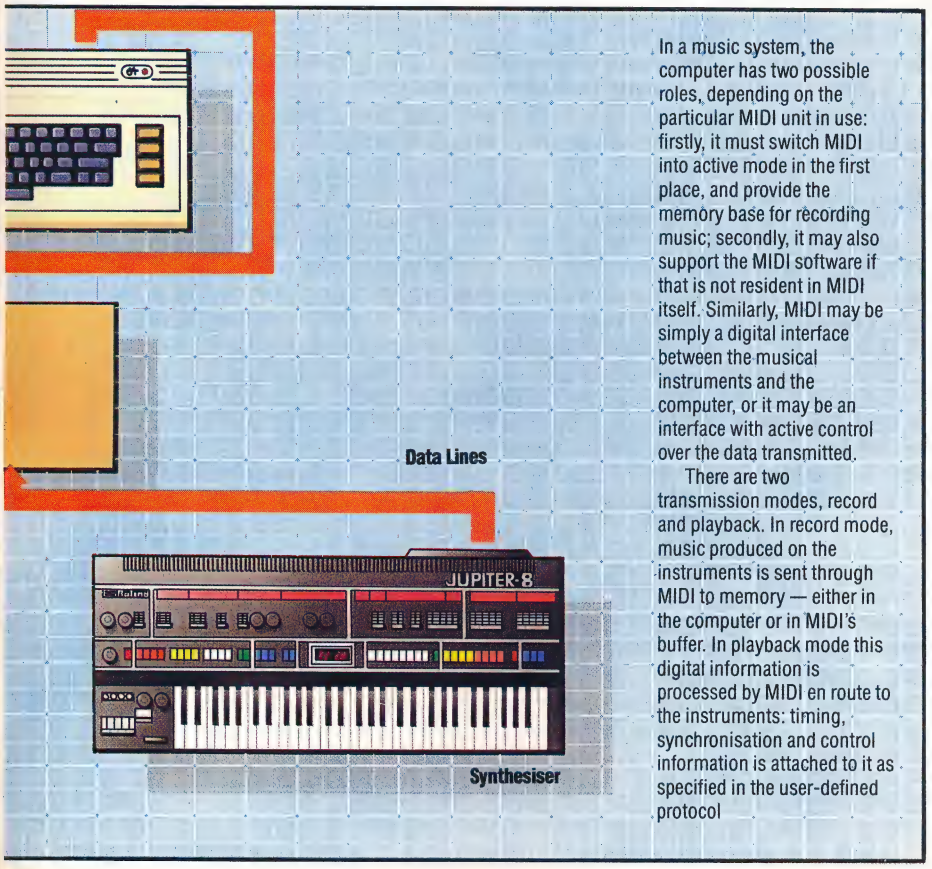


Gliss Bliss
The 'glissando' (bending the musical pitch from one note to another) is easily performed on a stringed instrument by sliding the fingers up or down the fretboard. A synthesiser needs careful programming to reproduce this as a 'slide' effect, and not as a series of distinct notes

MIDI differs from the RS232 interface design in an important respect. The rate of data transmission with the RS232 is 1,920 serial words per second, or 19.2 Kbaud: MIDI is half as fast again. This does not affect compatibility with home computers because logic circuitry within the MIDI itself sets a new clock speed of 3,125 words per second, or 31.25 Kbaud. This is a relatively high speed for serial transmission, but arguably it is not fast enough. We will consider why this may be so later in this article.

MIDI is designed to interface with more than a single receiving instrument. Where more than one instrument is receiving MIDI instructions, the first requirement must be that the correct data is sent to the appropriate instrument — otherwise a drum machine may end up trying to play a carefully sequenced melody, and a polyphonic synthesiser end up reproducing a bass drum pattern on its middle C. MIDI-compatible instruments are expected to have a numerical identification code or ID. One of the 16 available MIDI channels is assigned the code, so that only that channel accepts data for that instrument. The first part of a full MIDI transmission is then a status byte that includes the ID. All the data following this routing instruction can then specify how the command should be interpreted.

The unit, about 100 × 120 × 45 mm (4 × 5 × 2 in) in size, has two five-pin DIN sockets, marked 'MIDI IN' and 'MIDI OUT'. 'MIDI IN' accepts all instructions from a microcomputer or master synthesiser, and 'MIDI OUT' transmits the modified bitstream to the receiving instrument. Many models also have 'MIDI THRU', a second output socket that simply transmits the original unmodified bitstream sent to 'MIDI IN'. This may then be sent to a second interface. The cable, which has a maximum length of 15 metres, is fitted with five-pin DIN plugs, and connects to the back panel of a microcomputer or master synthesiser.



In a music system, the computer has two possible roles, depending on the particular MIDI unit in use: firstly, it must switch MIDI into active mode in the first place, and provide the memory base for recording music; secondly, it may also support the MIDI software if that is not resident in MIDI itself. Similarly, MIDI may be simply a digital interface between the musical instruments and the computer, or it may be an interface with active control over the data transmitted.

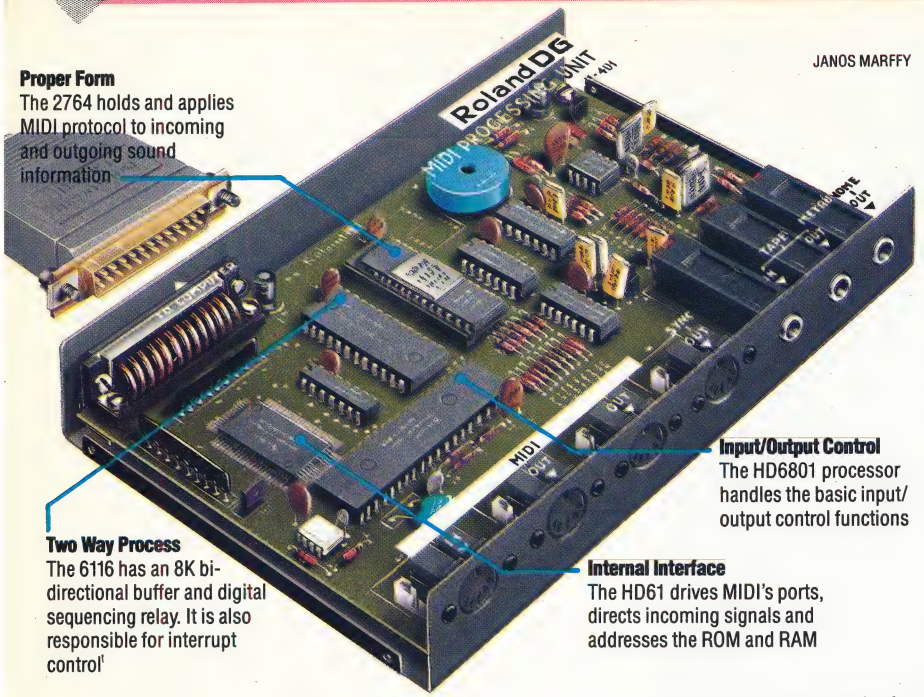
There are two transmission modes, record and playback. In record mode, music produced on the instruments is sent through MIDI to memory — either in the computer or in MIDI's buffer. In playback mode this digital information is processed by MIDI en route to the instruments: timing, synchronisation and control information is attached to it as specified in the user-defined protocol

KEVIN JONES

MIDDLE C

Let us imagine a first-time user of MIDI. He has a short melody he wishes to try out. The melody starts on middle C, moves upwards to E, then to G, and so on. How he instructs this will depend on the type of music software he is using. He may be using a light pen to dot in the notes on a five-line stave displayed on the VDU. This stave arrangement has been used as a standard notation format in Western music for over four centuries. He may be entering the information on the alphanumeric keyboard of his microcomputer, using some sort of MCL (Music Composition Language), again with a VDU display. Another alternative would be playing the notes of the melody on a music keyboard peripheral. This keyboard might have no sound of its own, but would produce one or other of the displays outlined above. But, however the music is entered, the MIDI transmission will always be the same: to start the tune, the first byte — transmitted as a serial word with its two extra bits from the ACIA chip — will instruct PLAY / ON CHANNEL 6; the second byte, MIDDLE C.

This minimal instruction will produce the note middle C from the receiving instrument. And the synthesiser will continue sounding middle C unless there is also an instruction to limit its duration, such as STOP PLAYING / ON CHANNEL 6, byte one; MIDDLE C, byte two; and ALLOWING FOR A DURATION OF X, byte three. If this second



JANOS MARFFY

Proper Form
The 2764 holds and applies MIDI protocol to incoming and outgoing sound information

Input/Output Control
The HD6801 processor handles the basic input/output control functions

Two Way Process
The 6116 has an 8K bi-directional buffer and digital sequencing relay. It is also responsible for interrupt control

Internal Interface
The HD61 drives MIDI's ports, directs incoming signals and addresses the ROM and RAM

The Musical Instrument Digital Interface
The MIDI interface reconciles the input/output protocols of the computer and musical instruments connected to it — just like any other interface — thus allowing the instruments to use the computer's memory. It also processes the digitised sound passed through it, adding control, sync and timing information to the synthesiser input

instruction is not given, and the rest of the melody, E, G, etc, is entered without duration parameters, all of the notes of the melody will continue to sound. The result will be a sustained chord made up of the notes of the melody.

Fortunately, even a slight acquaintance with the stave notation on the VDU will be enough to indicate that what was intended to be a melody seems likely to become a chord. And an MCL user who had made such an error could simply run the sequence, but this time send a MONOPHONIC instruction to the synthesiser. *Monophony* simply means one sound as opposed to many (*polyphony*). If the receiving synthesiser can produce only one sound at a time when set to MONOPHONIC, then the poorly-entered sequence can be performed as a succession of single notes only — a tune, rather than a chord.

Let us suppose that, after some trial and error, the first-time user has entered his tune accurately, and the interfaced synthesiser is now playing. The melody keeps time, and the rhythm — defined by the sequence of durations — is correct. It should be noted that so far in our discussion the types of instruction have been fairly limited. Only two musical parameters or characteristics have been called upon — pitch (middle C, E, G, etc.) and duration.

The composer of the melody listens to it a few times, then decides it sounds rather 'stiff' — as it is likely to do while it has only a minimum of definition. He decides that, instead of the first C and E occurring one after the other, the pitch of the C should glide upwards to the start of the E. This sort of movement is called a *glissando* or pitch bend, and would be characteristic of the way a person might whistle the tune. In this context, it might add a touch of jauntiness to the synthesiser's performance. So this instruction now replaces the original instruction for middle C, adding an extra byte.

This brings us to a simple point concerning MIDI interfacing. If the receiving synthesiser has no facility for producing *glissandi* (bending pitch), it cannot carry out this last instruction. It may perform the middle C as if it were receiving the *original* instruction, or it may do something else entirely. If a MIDI user's instructions are to produce a section of polyphonic music, and the receiving synthesiser is only a monophonic instrument, it will probably make an unpredictable selection from the polyphony, and then perform monophonically. In short, using MIDI to link a microcomputer to a very basic synthesiser will not turn it into an expensive synthesiser like the Fairlight.

These restrictions also apply in reverse. The receiving instrument may be a superb £10,000 synthesiser, but unless sufficient musical parameters have been defined, and unless the synthesiser's own controls have been set up as desired, the result may well be performed with the musicality of a pocket calculator.

In practice, the second of these two situations is easily improved. As many parameters as possible should be set as *constants* using the synthesiser controls, and the MIDI instructions should work within those parameters. This approach is the one most likely to be adopted by the synthesiser player whose problems we considered earlier.

So far, we have looked at pitch and duration characteristics, but MIDI provides for 128 theoretical controls, covering filtering, distortion, 'white noise' (all possible frequencies) and 'pink noise' (mid-range frequencies), each with values ranging from 0 to 128. This is more than adequate to deal with the parameters available on most synthesisers, and it is these controls that will probably interest microcomputer owners.

This is where the MIDI transmission rate becomes an issue. We have seen that a very straightforward command, concerning a single note and defining only two parameters, used three serial words. With the 31.25 Kbaud rate, this takes almost one millisecond. Six-note chords are common in many types of music: such a chord would take 5.76 milliseconds to transmit. If we now started to define this chord further using MIDI controls, the transmission time becomes slow enough for the human ear to begin to detect changes in the sound's characteristics caused by delay. These changes are apparent only when sounds, especially similar sounds, occur together — but as an audio interface MIDI was designed to handle simultaneous sounds. Music, unfortunately, is a 'parallel' medium: as listeners, we are used to hearing things happen simultaneously.

It is therefore not surprising that MIDI's serial transmission has been criticised; parallel transmission would have done the job better. It remains to be seen whether MIDI users are troubled by this failing. At present, the design of the interface appears to be a compromise between cost and efficiency, so it is worth remembering that the present specification may only be the first.

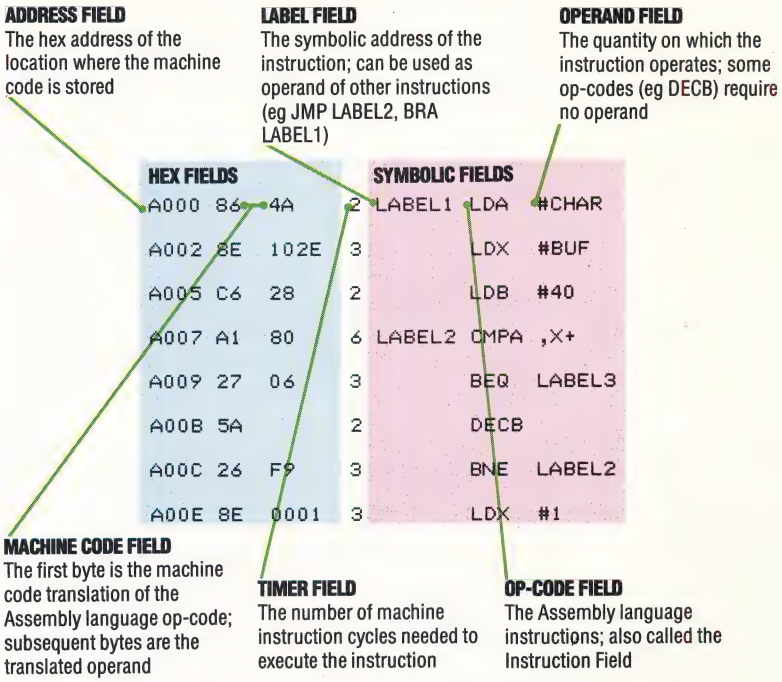
REGISTER TO REGISTER

Our introduction to 6809 machine code began with a general explanation of the role of the registers in the functioning of the microprocessor. Here we look at 6809 registers in greater detail and consider how they are used for storing and moving data.

We have seen that a register is a memory location within the processor chip itself, and have discussed how Assembly language programming involves manipulating the values stored in the registers and transferring these values to and from main memory. We further saw that some registers — mainly the ones that store and process addresses — are 16 bits in length, while others are eight bits in length, and that the various registers perform different functions.

- **Index registers** are used to modify the addresses that we use in our program.
- **Stack pointers** are used by the processor to address workspace memory and can be used in the same way by the programmer for quick storage and retrieval.
- The **program counter** holds the address of the next instruction, and can be altered by the programmer in order to transfer control, giving the Assembly language equivalent of a GOTO statement.
- **Accumulators** are the most frequently used registers, and are used to perform arithmetical functions.
- The **condition code** register contains a number of flags representing the state of the processor (such as whether the last operation gave a zero result); these flags can be tested in order to select or loop, giving the Assembly language equivalent of the IF...THEN structure.

The 6809 processor contains all these registers. As it is a development of the original Motorola 6800 processor (as is the 6502, which is used in the BBC Micro as well as many others), there are many similarities between the Assembly languages used on both processors. They are not, however, compatible — code written for one will not run on the other. Many 6800 programs are source code compatible, however — an Assembly language program written for the 6800 may be reassembled for the 6809 with at least a chance of it running. But even this small degree of compatibility is not available for the 6502 (or its later development, the 6510, which is used in the Commodore 64). However, the similarities between the processors at least mean that the task of translating an Assembly language program into a 6809 version is not too difficult and can be a good introduction to the 6809 for anyone familiar with the 6502. The 6809 contains the following registers:


ADDRESS FIELD
The hex address of the location where the machine code is stored
LABEL FIELD
The symbolic address of the instruction; can be used as operand of other instructions (eg JMP LABEL2, BRA LABEL1)
OPERAND FIELD
The quantity on which the instruction operates; some op-codes (eg DECB) require no operand
HEX FIELDS
SYMBOLIC FIELDS
A000 86 4A 2 LABEL1 LDA #CHAR
A002 8E 102E 3 LDX #BUF
A005 C6 28 2 LDB #40
A007 A1 80 6 LABEL2 CMPA ,X+
A009 27 06 3 BEQ LABEL3
A00B 5A 2 DECB
A00C 26 F9 3 BNE LABEL2
A00E 8E 0001 3 LDX #1
MACHINE CODE FIELD
The first byte is the machine code translation of the Assembly language op-code; subsequent bytes are the translated operand
TIMER FIELD
The number of machine instruction cycles needed to execute the instruction
OP-CODE FIELD
The Assembly language instructions; also called the Instruction Field

- Two eight-bit accumulators, known as A and B. There is no functional difference between them when used as eight-bit registers, so either may be used. The fact that there are two of them enables values to be retained in one accumulator while work is done in the other. Alternatively, the two eight-bit accumulators can be treated as a single 16-bit accumulator, a very powerful facility that allows the processor to carry out 16-bit arithmetic directly. Because of this, the 6809 is sometimes referred to as a pseudo-16-bit processor, but this is not really the case and it is better to think of it as being an advanced eight-bit processor. When the two accumulators are used together, they are referred to as the 16-bit D register.
- There are two index registers, referred to as X and Y. Again, there is no functional difference between them: either may be used in any situation. There is one slight operational difference, however, in that some instructions using the Y register will translate into two-byte instructions, as opposed to one byte for the corresponding X register instruction, thus making the program a little longer and slower. Where one index register only is required, it is therefore better to use X.
- The 6809 has two stack pointers, S and U. The processor uses S for all its stack operations. Although the programmer is free to use S if he or she wishes, it is always necessary to ensure that the

Fields Of Study
Assembly language programs look very different from BASIC program lists, and, when printed out by an Assembler program (which translates Assembly language into machine code) can seem completely baffling. The key to understanding them is to concentrate on the columns (or fields) that interest you (usually just the label, op-code and operand fields), and ignore the rest. The specimen program shown should help

operation of the processor is not affected by so doing — so it is safer to use U. The presence of two stack pointers makes the 6809 an ideal processor for use with FORTH.

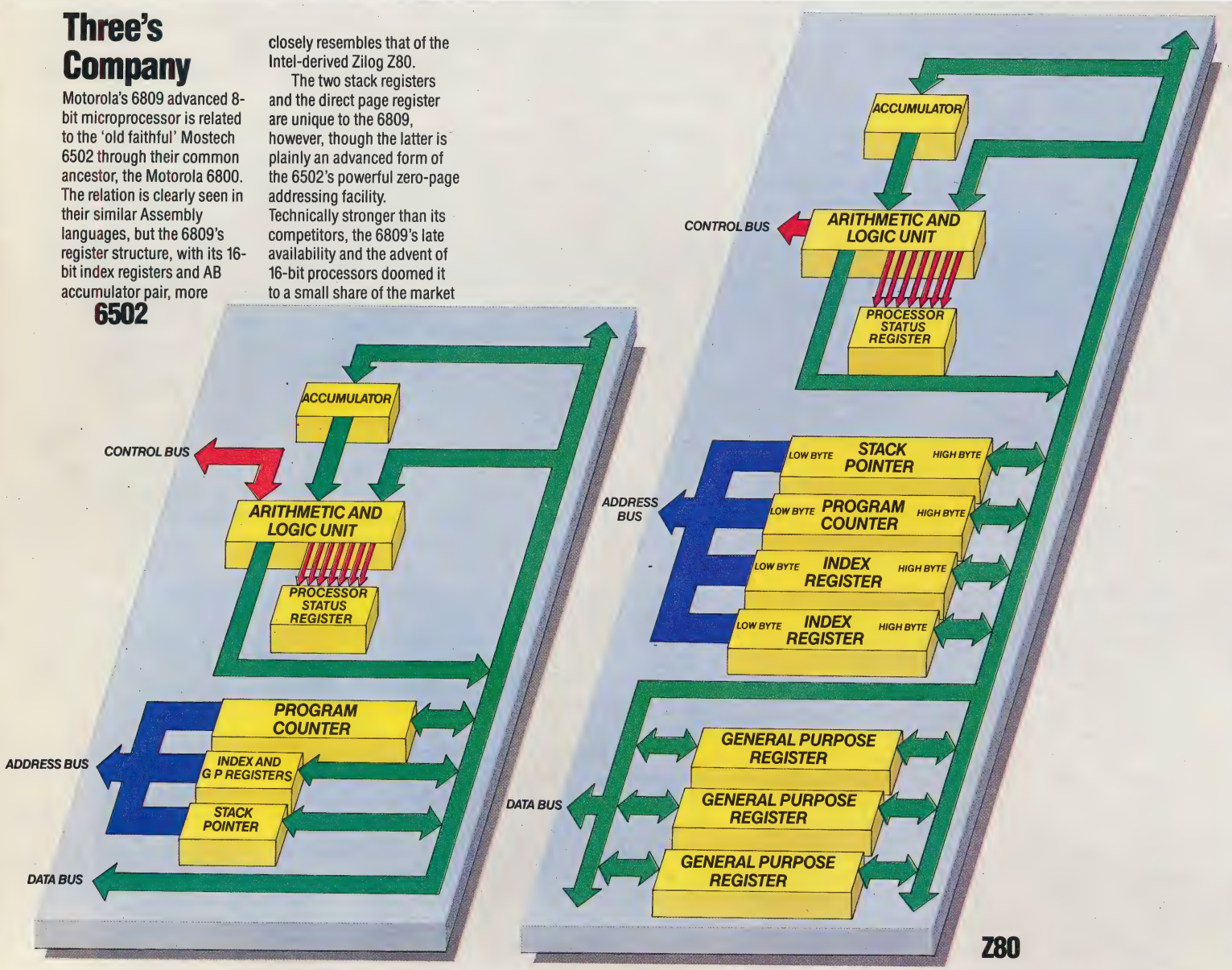
- The use of the X, Y, S and U registers is not restricted to their designated modes: both S and U may be used as index registers, for example, and all four registers can be used for storing and manipulating 16-bit numbers.
- The program counter (PC) is a 16-bit register that is automatically adjusted by the processor so that it points to the next instruction. The jump (JMP) and branch (BRA) instructions alter the contents of the program counter and the 6809 allows it to be used as a type of index register. We shall look more closely at the effects of this later in the course, but by using addresses relative to the PC contents instead of specifying some absolute address stored in X or Y, the programmer can write true position-independent code. In other words, properly written programs can be loaded unaltered into any position in memory and run from there.
- The condition code register (CC) has eight bits that are used independently as flags to signal the condition of the processor. Instructions such as branch not equal (BNE) test an individual flag and cause a change in the flow of control according to the flag's condition (one or zero).

There are several instructions that are used simply for moving data into, out of and between the various registers; for the purposes of the following examples, let us suppose that we have reserved a number of memory locations by using the Assembler directives FCB and FDB. These are not instructions to the processor. When the Assembler translates the Assembly language program into machine code, it will obey them immediately and set aside the desired memory locations for program use. It will not translate the directives into machine code because they do not concern the processor. As they resemble Assembly language operation codes (such as BNE or JMP) they are sometimes called pseudo-ops;

Three's Company

Motorola's 6809 advanced 8-bit microprocessor is related to the 'old faithful' Mostech 6502 through their common ancestor, the Motorola 6800. The relation is clearly seen in their similar Assembly languages, but the 6809's register structure, with its 16-bit index registers and AB accumulator pair, more closely resembles that of the Intel-derived Zilog Z80.

The two stack registers and the direct page register are unique to the 6809, however, though the latter is plainly an advanced form of the 6502's powerful zero-page addressing facility. Technically stronger than its competitors, the 6809's late availability and the advent of 16-bit processors doomed it to a small share of the market

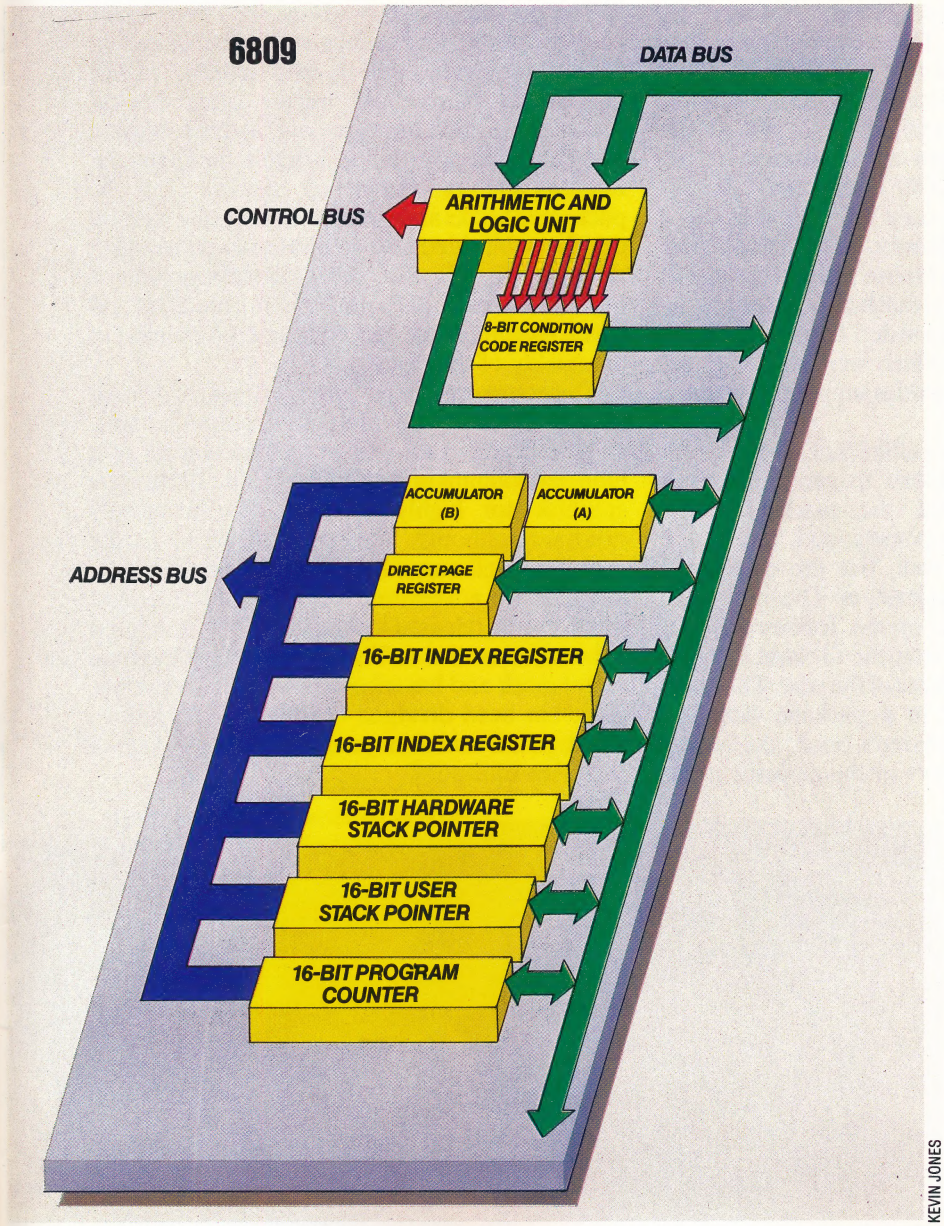


'Assembler directives' is a better name as this explains their function — to direct the functioning of the Assembler program. If these directives are labelled, then the label will be translated by the Assembler into the appropriate address — so we may have, for example:

```
NUM1 FCB 0      reserving a single byte that will
                be referred to as NUM1, with
                initial value 0
NUM2 FCB 0      similar to the above
NUM3 FDB #A93B  reserves two bytes for the 16-
                bit number #A93B (#, the
                'hash' sign, is often used by
                6809 Assemblers as a sign
                that the number is in
                hexadecimal notation)
```

The following instructions load the values stored in these locations into various registers:

```
LDA NUM1   will load the eight-bit number stored at
           the memory location represented by
           NUM1 into accumulator A
LDB NUM2   as above, loads NUM2 into
           accumulator B
LDX NUM3
LDY NUM3   These instructions will load the 16-bit
LDS NUM3   number in NUM3 into the X, Y, S, U and
LDU NUM3   D registers respectively
LDD NUM3
```



KEVIN JONES

In a similar way, the eight- or 16-bit contents of a register may be stored in a memory location by using one of:

```
STA NUM1
STB NUM2
STX NUM3
STY NUM3
STS NUM3
STU NUM3
STD NUM3
```

Notice that when the accumulator is loaded from NUM1, you actually copy NUM1 into the accumulator without changing it; the store operations function similarly.

The contents of two registers may be exchanged (provided that they are the same size) by using the EXG instruction. For example:

```
EXG A,B  exchanges the contents of registers A and B
EXG X,S  exchanges the contents of registers X and S
```

The contents of one may be transferred to another — for example: TFR Y,U copies the contents of Y into U. To accomplish this, the two registers must again be of the same size, both eight-bit or both 16-bit.

In order to write a program that actually does something, let us introduce the ADD instruction, which will add the contents of a memory location to the contents of one of the accumulators. It takes the form:

```
ADDA NUM1  meaning 'add the contents of memory
           location NUM1 into the A register,
           leaving the A register containing the
           result of the addition'
```

First we will add the two eight-bit numbers in NUM1 and NUM2, putting the answer back in NUM1 and ignoring any overflow if their sum is larger than an eight-bit number. We will then add the two locations' contents again, but this time obtaining a 16-bit result in NUM3.

First example:

```
LDA NUM1   copy first number into A
ADDA NUM2  add second number
STA NUM1   store answer back in NUM1
```

Second example:

```
LDB NUM1   copy first number into B
SEX        convert the eight-bit number in B into
           a 16-bit number in D
STD NUM3   copy D into NUM3
LDB NUM2   copy second number into B
SEX        convert it to 16-bit number in D
ADDD NUM3  add the first 16-bit number from
           NUM3 into D
STD NUM3   store the answer back in NUM3
```

FOSTERING AN IMAGE

Melbourne House is a company that is best known for its adventure software, including games such as The Hobbit and Mugsy, which attract attention for their high standard of graphics and well-crafted plots. But the company also produces books, and its publications are invaluable aids to home computer programmers.

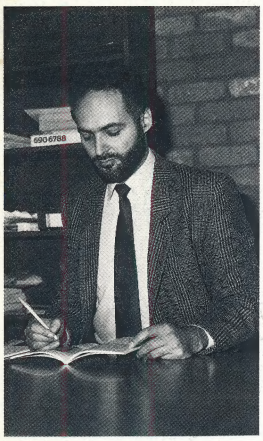
Melbourne House was founded in 1977 by an Australian, Fred Milgrom. The launch of the Sinclair ZX80 alerted Milgrom to the potential rewards of publishing books on home computing, and in 1980 Melbourne House produced *30 Programs for the ZX80*. The success of this led to a series of books devoted to the Sinclair machine, and the company released its first software package — Space Invaders, again for the ZX80.

The following year saw the appearance of the ZX81. Demand for ZX80 books and cassettes plummeted, and it was only US sales that saved the company from disaster. The lesson was learned and Melbourne House realised the merits of diversification. As new machines came onto the market, the company provided users with books and software, sales of which were aided by the poor quality of the user guides supplied with some home computers.

The immediate success of the Sinclair Spectrum enabled Melbourne House to produce games software that exploited the machine's high resolution colour graphics and sound. The arcade game Penetrator sold well, but the company's biggest coup was the release of The Hobbit, a graphic adventure based on the Tolkien novel of the same name, which won the Golden Joystick award for best strategy game of the year. The game cassette was marketed in a package that also contained a copy of Tolkien's book; this was a condition set by executors of the author's estate and resulted in The Hobbit being sold at a price three times as high as most Spectrum software then available. Despite the cost, sales were extremely good and The Hobbit is now available on other home machines, including the BBC Micro and Oric/Atmos.

The in-house programmers are based in Melbourne, Australia. Each team of four concentrates on one aspect of a game. This means that more time is spent in game development, but the policy has brought high sales and customer loyalty. The company hopes to capitalise on this loyalty in a drive to sell more books. Paula Byrne, the company's publicity manager, points out: 'When people go to buy a book they don't know what they want, and so they will probably end up buying something that is not suitable, which discourages them from buying other books.' Melbourne House hopes to combat this confusion by marking their books as suitable for Beginner, Intermediate or Advanced readers and by packaging books and software in the same style, so that the public identifies quality books with quality software. To this end, the company includes a registration card with each of its products, on which customers are invited to give an opinion of the quality of their purchase.

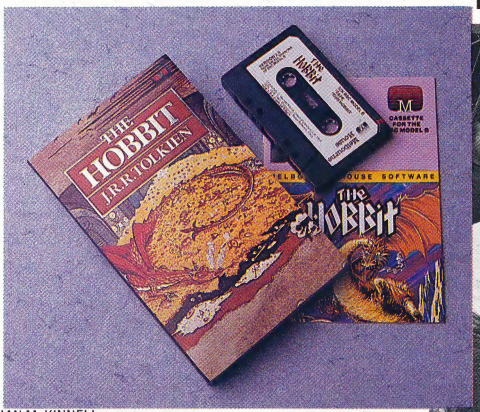
The company's latest game, Mugsy, is claimed to be the 'world's first interactive computer comic strip' and allows the player to take control of a mob of hoodlums in the Chicago of the 1920s. The graphics are highly detailed and beautifully designed, although the game itself is hardly complex. Melbourne House is also working on a Hobbit-style adventure called Sherlock Holmes, which will be released in late 1984. This is reputed to be as innovative as The Hobbit was at the time of its launch and has taken 15 months to develop. Little has been divulged of the game's contents, although it is said to require a good knowledge of Victorian transport!



Alfred Milgrom, Co-Director and Publisher, Melbourne House

Philip Mitchell, author of The Hobbit game and Sherlock Holmes

Book Of The Game
In addition to creating a fun and exciting game in The Hobbit, Melbourne House also includes a copy of JRR Tolkein's masterpiece in the package, a brilliant marketing ploy



IAN McKINNELL

Part of the Melbourne House software team

The Workshop Answers Back

The programming exercises set in the last instalment (see page 516) all resembled the Telephone Numbers example program, in monitoring the user port and checking its state against that of some reference value. We give workable, but by no means unique, solutions.

1) Burglar Alarm

```
 10 REM BBC VERSION 1.1
 20 DATREG=&FE60:DDR=&FE62
 30 ?DDR=0:?DATREG=127
 40 REPEAT
 50 PRINT "ALL'S WELL"
 60 ALARM=?DATREG
 70 UNTIL ALARM<>255
 80 PRINT "ALARM"
 90 FOR PIN=0 TO 7
100 IF (ALARM AND 2^PIN)=0
THEN PRINT "ALARM ON SWITCH NO."PIN
110 NEXT
120 END
```

```
10 REM CBM64 VERSION 1.1
20 DDR=56579: DATREG=56577
30 POKE DDR,0: POKE DATREG,255
40 FOR C=0 TO 1 STEP 0
50 PRINT "ALL'S WELL",
60 ALARM=PEEK (DATREG)
70 IF ALARM<>255 THEN C=1
80 NEXT C
90 PRINT "ALARM SOUNDS"
100 FOR PIN=0 TO 7
110 IF (ALARM AND 2↑PIN)=0 THEN PRINT"ALARM ON SWITCH #";PIN
120 NEXT:END
READY.
```

2) Pulse Count (i)

```
 10 REM BBC VERSION 1.2
 20 DATREG=&FE60:DDR=&FE62
 30 ?DDR=0:?DATREG=127
 40 KOUNT=0: REM INIT COUNT
 50 TIME=0: REM INIT TIME
 60 REPEAT
 70 REPEAT
 80 UNTIL ?DATREG<>255
 90 KOUNT=KOUNT+1
100 UNTIL TIME >6000: REM 1 MIN
110 PRINT "PULSES="KOUNT
120 END
```

```
10 REM CBM64 VERSION 1.2
20 DDR=56579:DATREG=56577
30 POKE DDR,0: REM ALL INPUT
40 T=TI: REM INIT TIME
50 IF PEEK(DATREG)=255 THEN 50
60 COUNT=COUNT+1
70 IF (TI-T)<3600 THEN 50
80 PRINT "PULSES=";COUNT
90 END
READY.
```

3) Pulse Count (ii)

```
 10 REM BBC VERSION 1.3
 20 DATREG=&FE60:DDR=&FE62
 30 DIM C(8)
 40 ?DDR=0: TIME=0
 50 REPEAT
 60 FOR N=0 TO 7
 70 C(N)=C(N)+NOT(?DATREG AND(2^N))
 80 NEXT N
 90 UNTIL TIME>6000
100 FOR N=0 TO 7
110 PRINT "LINE"N"="C(N)
120 NEXT N
130 END
```

```
10 REM CBM64 VERSION 1.3
20 DDR=56579:DATREG=56577
30 POKE DDR,0:T=TI
40 FOR K=0 TO 7
50 C(K)=C(K)+NOT(PEEK(DATREG)AND(2↑K))
60 NEXT K
70 IF (TI-T)<3600 THEN 40
80 FOR K=0 TO 7
90 PRINT "LINE";K;"="C(K)
100 NEXT K
110 END
```

4) Combination Lock

```
 10 REM BBC VERSION 1.4
 20 DATREG=&FE60:DDR=&FE62:FLAG=1
 30 ?DDR=15: REM LINES 0-3 INPUT
 40 DIM C(3)
 50 REM CODE=359
 60 C(1)=3:C(2)=5:C(3)=9
 70 FOR N=1 TO 3
 80 A$=GET$: REM AWAIT KEYPRESS
 90 NEXT N
100 IF FLAG=1 THEN PRINT "UNLOCKED"
ELSE PRINT "WRONG COMBINATION"
110 END
```

```
10 REM CBM64 VERSION 1.4
20 DDR=56579:DATREG=56577
30 POKE DDR,15: REM LINES 0-3 INPUT
40 C(1)=1:C(2)=2:C(3)=4: REM CODE 124
50 FOR K=1 TO 3
55 PRINT"ENTER A DIGIT"
60 GET A$: IF A$="" THEN 60: REM AWAIT KEY
65 PRINT A$
70 IF PEEK(DATREG)<>C(K) THEN FL=1
80 NEXT K
90 IF FL=1 THEN PRINT "WRONG COMBINATION": END
100 PRINT "UNLOCKED"
110 END
```

5) Change Colour

```
 10 REM BBC VERSION 1.5
 20 MODE5
 30 DATREG=&FE60:DDR=&FE62
 40 ?DDR=128: REM ALL INPUT BUT D7
 45 ?DATREG=255
 50 A$=GET$: REM AWAIT KEYPRESS
 60 SCOL=?DATREG: REM BACKGROUNDS>127
 70 GCOL0,SCOL: REM CHANGE SCREEN COLOUR
 80 CLG: REM CLEAR GRAPHICS SCREEN
 90 END
```

```
10 REM CBM64 VERSION 1.5
20 DDR=56579: DATREG=56577
30 POKE DDR,0: REM ALL INPUT
40 GET A$: IF A$="" THEN 40
50 SCOL=PEEK(DATREG)
60 POKE 53281,SCOL
70 END
```

IMAGES PRODUCED ON THE ARTRON 2000 STUDIO COMPUTER; COURTESY OF GRAPHIC PRODUCTS, LONDON (EXCLUSIVE DISTRIBUTOR)